ciencia popular

A. I. Saltykov
G. L. Semashko



Programación para todos



```
PROGRAM PAL
PRINT 10
FORMAT (10X,`*
C8X,`*******`/8X
C14 X, 13(`*`),1X,
```

Este es un Curso Elemental para programación de ordenadores en Fortran que proporciona las nociones básicas, para que cualquiera que posea o tenga acceso a un ordenador, pueda programar.

Editorial·Mir·Moscú

PROGRAMACIÓN PARA TODOS

А. И. САЛТЫКОВ. Г. Л. СЕМАШКО

ПРОГРАММИРОВАНИЕ ДЛЯ ВСЕХ

ИЗДАТЕЛЬСТВО «НАУКА»
МОСКВА

A. I. Saltykov, G. L. Semashko

Programación para todos



EDITORIAL MIR MOSCÚ

Traducido del ruso por el ingeniero
Virgilio Llanos Más

Impreso en la URSS

На испанском языке

ISBN 5-03-000658-3

Índice

Prefacio

En el libro se interpretan las cuestiones relacionadas con la programación en los ordenadores.

El libro se compone de cuatro capítulos. El capítulo I está destinado para el lector que por primera vez comienza a programar en un ordenador. Aquí se dan las ideas elementales respecto a la programación, a la estructura de los ordenadores, a los lenguajes algorítmicos. Más adelante el lector hace conocimiento con los elementos del lenguaje Fortran, uno de los lenguajes algorítmicos más difundidos y simples.

El lenguaje Fortran, a pesar de su edad venerable, sigue siendo todavía uno de los lenguajes de programación más difundidos para la solución de problemas de cálculo de la ciencia y de la técnica. Siendo suficientemente simple para el estudio, este lenguaje puede ser útil también para el dominio ulterior de lenguajes de programación más modernos (Pascal, Ada y otros). Señalaremos que la propiedad de modulación, que por primera vez fue realizada en el Fortran, se trasladó en uno u otro aspecto a muchos lenguajes de programación creados más tarde. Para comodidad de los lectores los párrafos dedicados a la descripción

del lenguaje Fortran van acompañados de una enumeración detallada de los puntos. El círculo de cuestiones enfocadas en el capítulo I no sólo permite escribir los programas más simples, sino también obtener en el ordenador los resultados de su trabajo.

Puesto que el material de este capítulo está destinado para el lector principiante, en una serie de casos nos vimos obligados a renunciar de la rigurosidad de redacción. La descripción estricta de las nociones que se introducen es poco abordable para el lector principiante. Por eso adoptamos la siguiente metodología de exposición. En la primera etapa se dan formulaciones que, por un lado, son comprensibles al lector principiante y, por otro lado, son suficientemente correctas teniendo en cuenta su precisión ulterior. Durante la exposición sucesiva del material se comunican conocimientos complementarios, que permiten al lector formar una idea más completa sobre la noción dada.

En el capítulo II se expone una serie de cuestiones destinadas para el lector poseedor de conocimientos en el conjunto del primer capítulo. Aquí se exponen nuevos conocimientos sobre el Fortran, necesarios para programaciones bastante complicadas.

Al programador que trabaje con un ordenador le pueden ser útiles los conocimientos aquí expuestos sobre el funcionamiento del compilador, y asimismo los ejemplos alec-

cionadores, que contienen recomendaciones prácticas para la realización de ciertos algoritmos numéricos. En el capítulo III se da a conocer una información respecto a las versiones y estándares del lenguaje Fortran, provechosa para el desarrollo general.

En el capítulo IV se exponen ejemplos de programas simples.

A algunos lectores les puede parecer que en el libro se estudian frecuentemente cuestiones de carácter técnico demasiado «elementales». Nuestra experiencia muestra que cuando el usuario comienza un nuevo tipo de trabajo en el ordenador le surge una multitud de preguntas de este tipo. Por ejemplo, uno se prepara para poner en marcha, por primera vez, un programa en un ordenador. ¿Qué debe hacer para lograrlo? O, digamos, en el programa se prevé un trabajo con la memoria externa. ¿Cómo realizarlo prácticamente? Muchos usuarios necesitan estos conocimientos elementales. Los programadores principiantes gastan mucho tiempo y fuerzas en búsqueda de información de este género. Nosotros nos esforzamos para ayudarlos con recomendaciones concretas.

Los autores

CAPÍTULO I

Para los principiantes

1.1. El ordenador en nuestra vida

Desde tiempos antiguos el hombre creó un sinnúmero de mecanismos y máquinas de toda clase para facilitar su trabajo. No obstante, hasta hace poco tiempo, las máquinas facilitaban solamente el trabajo físico del hombre. La actividad relacionada con el trabajo intelectual transcurría sin una participación seria de las máquinas.

La creación de ordenadores que son sumamente rápidos se considera justamente como uno de los logros científico-técnicos notables de la humanidad.

Los ordenadores modernos son capaces de realizar en un segundo varios millones de operaciones matemáticas (sumas, restas, etc.), mientras que una persona experimentada necesita medio minuto, por término medio, para realizar una operación aritmética. De este modo, con la aparición de los ordenadores, en un espacio breve de tiempo (unos 35 años) la velocidad de los cálculos aumentó aproximadamente en 100 millones de veces.

Citaremos un ejemplo que ilustra brillantemente las ventajas de los ordenadores.

El matemático inglés del siglo XIX W. Shenks gastó más de 20 años de su vida para calcular el número π con precisión de 707 cifras significativas exactas. Este resultado cobró la fama del record de los cálculos del siglo XIX. Sin embargo, más tarde se descubrió que W. Shenks se había equivocado en el 520-avo signo, por lo que todas las cifras significativas consecuentes fueron calculadas erróneamente.

En el ordenador el número π ha sido calculado con precisión de 500 mil signos. Para ello se necesitaron tan sólo algunas horas de funcionamiento de la máquina.

Semejantes cualidades de los ordenadores como son la velocidad operacional y la posibilidad de obtener un resultado correcto garantizado, han permitido resolver problemas nuevos en principio. Por ejemplo, antes de la aparición de los ordenadores el problema de control de un cohete desde la Tierra, en principio, no pudo ser resuelto. El volumen de los cálculos es tan grande que las coordenadas de un solo punto de la trayectoria del cohete, si el cálculo fuese manual, se obtendrían después de la caída de éste.

Los ordenadores se hicieron habituales en nuestra vida. Muchos de nosotros, partiendo en tren en un viaje largo, utilizamos los servicios de los ordenadores. Los billetes que se adquieren son expedidos directamente desde un teletipo controlado por un ordenador.

Los ordenadores calculan los elementos de la órbita del satélite artificial de la Tie-

rra y controlan los procesos de producción en la fábrica, ayudan a los árbitros en las competiciones deportivas y administran la distribución de los asientos en los aviones. Ya en 1973 los empleados de la revista norteamericana «Computers and Automation» registraron 2500 esferas de aplicación de los ordenadores.

Para poder resolver diversos problemas de la ciencia y la economía nacional, se crean ordenadores de distintos tipos. Unos de ellos tienen un destino específico (por ejemplo, para el control de una nave cósmica). Estos ordenadores se denominan especializados. Otros, denominados universales, pueden emplearse en diferentes ramas de la ciencia y de la técnica. Todo lo que se dice en este libro concierne solamente a los ordenadores universales, los que llamaremos en lo sucesivo simplemente ordenadores, o máquinas.

1.2. ¿Para qué necesita la máquina el programa?

En muchos cuentos orientales se narra sobre los poderosos dzinnes *) capaces de realizar milagros. El dzinn sirve sumisamente a cada uno que sepa los conjuros correspondientes y pueda pronunciarlos bien. Pero si el hombre olvidó o confundió el conjuro el dzinn se niega a servirle.

*) Dzinn (del árabe), genio, espíritu, demonio en los cuentos árabes y persas. (N. del T.)

Los ordenadores modernos, en cierto sentido, se comportan igual que los dzinnes. Para ellos los «conjuros» son los programas, elaborados por las personas (por los programadores). Las máquinas cumplen exactamente todas las operaciones ordenadas por los programas. Estos últimos se escriben en conformidad rigurosa con determinadas reglas. La mínima infracción de estas reglas puede llevar a que la máquina se niegue a ejecutar las operaciones necesarias. Sólo cumpliendo estrictamente todas las reglas formales de escritura de programas se puede obligar al ordenador a servir lealmente.

Lo mismo que los dzinnes cumplían con igual disposición la voluntad de su señor, tanto buena como mala, las máquinas siguen también pedantemente las indicaciones de cualquier programa en el que no existen infracciones de las reglas formales. La máquina solamente será «inteligente» cuando sea «inteligente» el programa introducido en ella: el programador eligió un método acertado de solución del problema y lo realizó racionalmente en el ordenador.

Debemos subrayar que la máquina no puede notar los errores en el algoritmo del programa (por ejemplo, en lugar de adición se ha programado la sustracción).

El programador debe pensar bien y escribir inequívocamente la secuencia de operaciones que conduce a la solución del problema. El hecho es que la estructura de la máquina no permite ejecutar directamente

una operación bastante complicada, por ejemplo, resolver una ecuación cuadrática.

El ordenador está hecho de tal forma que solamente sabe ejecutar operaciones aritméticas y algunas otras operaciones simples. El hombre determina el programa de funcionamiento de la máquina. En el programa, que se compone de instrucciones separadas, se indica qué operaciones, en qué orden y con qué números debe realizarse.

Por ejemplo, para calcular en un ordenador el área de un círculo con radio $R = 15$, a la máquina se le dan las instrucciones siguientes.

1. Multiplicar 15 por 15.
2. Multiplicar el resultado anterior por 3,14159.
3. Imprimir el resultado.
4. Terminar el trabajo («stop»).

Para resolver este problema la máquina ejecutará dos operaciones de multiplicación y cumplirá la instrucción de imprimir. Al recibir la orden de «stop» la máquina dirigirá su «atención» a un programa nuevo.

Estudiemos qué operaciones debe realizar la máquina para resolver la ecuación lineal $ax = b$. Recordemos cómo se resuelven tales ecuaciones. Si $a \neq 0$ existe una única solución $x = b/a$; si $a = 0$ y $b = 0$ la solución es cualquier valor de x; si $a = 0$, y $b \neq 0$, la ecuación no tiene solución.

Escribamos ahora el programa de cálculo como la secuencia de siguientes instrucciones:

1. Comprobar si a es igual a cero. Si lo es pasar al punto 5, y si no lo es, continuar los cálculos.

2. Dividir b por a.

3. Imprimir el resultado.

4. Terminar el trabajo («stop»).

5. Comprobar si b es igual a 0. Si lo es, pasar al punto 8, y si no, continuar los cálculos.

6. Imprimir el texto: «La ecuación no tiene solución».

7. Terminar el trabajo («stop»).

8. Imprimir el texto: «x es cualquier número».

9. Terminar el trabajo («stop»).

En dependencia de cuáles sean los valores de a y b, la máquina ejecutará las instrucciones 1, 2, 3, 4 ($a \neq 0$), o 1, 5, 6, 7 ($a = 0$, $b \neq 0$), o bien 1, 5, 8, 9 ($a = 0$, $b = 0$).

En el ejemplo expuesto el programa prevé el caso $a = 0$. Si no se quiere realizar la investigación respecto a la posibilidad de que a y b sean iguales a cero, el programa será entonces mucho más corto.

1. Dividir b por a.

2. Imprimir el resultado.

3. Terminar el trabajo («stop»).

Pero, en este caso, cuando resulte ser que $a = 0$, la máquina interrumpirá la ejecución del programa al intentar dividir por cero. El programador no sabrá cuál de los dos casos tuvo lugar: o el problema no tiene solución o tiene un conjunto infinito de soluciones.

En todos los casos cuando la máquina no puede realizar cualquier operación (dividir por cero, extraer la raíz de un número negativo, multiplicar dos números demasiado grandes, etc.) ella interrumpe la ejecución del programa. Semejantes situaciones, como regla, son la consecuencia de errores cometidos en la confección del programa, que se deben encontrar y corregir.

Así, al resolver cualquier problema, el ordenador cumple un programa, en el que se describe exactamente todo el curso de los cálculos.

La diversidad de los problemas que se resuelven con la máquina está basada en el hecho de que, mediante los programas, se pueden dictar diferentes secuencias de las operaciones y distintos datos iniciales.

Las cuestiones relacionadas con la metodología de confección de programas para los ordenadores exigieron la creación de toda una disciplina científica aplicada denominada programación.

Como regla, pueden componerse varios programas distintos para un mismo fin. Estos programas se diferenciarán unos de otros por su longitud y velocidad de ejecución en la máquina. Para la escritura de un programa «bello» (es decir, óptimo por sus parámetros) se deben conocer bien la programación, las matemáticas, los métodos de cálculo, mostrar ingeniosidad y esforzarse por crear un programa perfecto.

Antes de llegar a ser un músico de verdad es necesario practicar obstinadamen-

te durante largo tiempo la técnica de interpretación, tocando bosquejos y ejercicios aburridos. Así mismo el programador debe comenzar por el estudio tenaz y la memorización de las reglas fundamentales de la programación, y solamente después de esto podrá experimentar la alegría del trabajo creador.

1.3. Anatomía del ordenador

Si se propone a una persona resolver cualquier problema ésta necesita

a) leer la condición del problema,

b) retenerla en la memoria,

c) realizar las operaciones necesarias que conducen a la solución del problema,

d) comunicar el resultado al «cliente».

El ordenador realiza operaciones análogas: «lee», «retiene en la memoria», realiza operaciones, «comunica» el resultado al hombre.

Todos estos procesos transcurren en dispositivos especiales del ordenador.

1.3.1. Unidad de entrada

La máquina «lee» el programa mediante el dispositivo o unidad de entrada. Uno de los procedimientos de entrada de información más difundido es el de entrada con fichas perforadas.

Existen dispositivos especiales, denominados perforadoras, que están destinados a

perforar las tarjetas. A cada letra y cifra les corresponde una combinación de orificios.

Después de la perforación todo el programa representa en sí un paquete de fichas perforadas. Este paquete se introduce en el dispositivo de lectura y la máquina «lee» ficha por ficha todo el programa. Durante el proceso de lectura la combinación de orificios en la ficha se transforma en señales eléctricas. La existencia de señal corresponde a la cifra «1», y la ausencia, a la cifra «0». Esta transformación transcurre, por ejemplo, mediante una fotocélula: la existencia del orificio conduce a la respuesta de la fotocélula sobre la que cayó el rayo de luz; si no existe el orificio la fotocélula no responde y no habrá señal alguna.

De este modo, cada ficha perforada que pasó por la máquina engendra cierta combinación de ceros y unidades.

El programa también puede introducirse desde el teclado del display: cada opresión de una tecla conduce a la aparición en la máquina de una serie de señales eléctricas, lo que corresponde a una composición de ceros y unidades.

1.3.2. Memoria del ordenador

Una vez leído el programa, la máquina lo «retiene» en su memoria. La capacidad de la memoria del ordenador frecuentemente se mide por el número de células. En una célula se puede almacenar un número o una instrucción del programa.

La unidad de medida de la capacidad de memoria se llama el *byte*. Cada byte puede almacenar un símbolo de información textual (letra, cifra, signo). Durante los cálculos varios bytes se unen en una palabra (célula) del ordenador.

La capacidad de memoria de los ordenadores modernos es mayor de un megabyte. Por ejemplo, la EC-1061 tiene una capacidad de 8 megabytes.

Cada símbolo se registra con una combinación de varios ceros y unidades: cada orden binario se denomina *bit* de información. Un byte se compone de 8 bits.

La memoria de la que tratamos se denomina operativa: durante la ejecución de cualquier operación (por ejemplo, adición) la máquina elige en esta memoria los números (sumandos) y envía el resultado allí mismo.

El ordenador, además de la memoria operativa, tiene también memoria externa: cintas, discos y tambores magnéticos. La información almacenada en la memoria externa no puede ser utilizada directamente: debe ser previamente copiada en la memoria operativa. La memoria externa supera a la operativa en centenares de veces, no obstante, la copia de la información es una operación relativamente lenta: esto debe tenerse en cuenta al componer programas.

1.3.3. Dispositivo aritmético y lógico

Las operaciones como tales (suma, resta, multiplicación, división y algunas otras) se realizan en el dispositivo aritmético y lógico. En él, desde la memoria, ingresan aquellos números con los que se debe realizar la operación en turno. El resultado obtenido o bien queda en el dispositivo aritmético o bien se inscribe inmediatamente en la memoria.

1.3.4. Unidad de salida

Después de que la máquina resuelve el problema y obtiene el resultado, éste se pasa a la unidad de salida. Esta unidad transforma las señales eléctricas en cifras, letras, gráficos. El procedimiento más difundido de salida es la impresión de los resultados en una banda ancha de papel (listing).

A muchos ordenadores se les conectan registradores gráficos, y los resultados del cálculo pueden obtenerse en forma de gráficos.

Actualmente obtienen una mayor difusión los displays: dispositivos que tienen pantalla de televisión y teclado.

Los displays sirven tanto para la salida como para la entrada de información. Semejantes dispositivos, instalados a cierta distancia del ordenador y conectados a él por una línea de cable de telecomunicaciones, se denominan terminales.

1.3.5. *Dispositivo de control*

Todo el funcionamiento de la máquina lo organiza el dispositivo de control. Éste realiza el descifre de cada instrucción y envía las señales a otros dispositivos del ordenador que participan en la ejecución de la instrucción.

Así, los dispositivos principales del ordenador electrónico son:

1. unidad de entrada,
2. memoria,
3. dispositivo aritmético y lógico,
4. unidad de salida,
5. dispositivo de control.

Los dispositivos aritmético y lógico generalmente se acoplan al dispositivo de control. Semejante dispositivo se denomina unidad de procesamiento (cálculo), o simplemente procesador.

1.4. Noción de algoritmo

El significado actual de la palabra «algoritmo» es parecido al de las palabras: receta, procedimiento, método.

El algoritmo es el conjunto de un número finito de instrucciones que determinan el orden de ejecución de las operaciones al resolver el problema.

He aquí algunos ejemplos de composición de algoritmos de solución de ciertos problemas.

Problema 1. Ir de la ciudad de Dubná a Moscú.

Algoritmo de solución

1. Venir a la estación «Dubná».
2. Comprar el billete hasta Moscú.
3. Si está allí el tren sentarse en él.
4. Bajar en la estación de Moscú.

Problema 2. Resolver la ecuación $2x^2 - 3x - 5 = 0$.

Algoritmo de solución

1. Calcular el discriminante $D = 3^2 + 4 \times 2 \times 5$.
2. Extraer la raíz de D.
3. Calcular $x_1 = \frac{3 + \sqrt{D}}{2 \times 2}$.
4. Calcular $x_2 = \frac{3 - \sqrt{D}}{2 \times 2}$.
5. Escribir la respuesta.
6. Terminar el trabajo.

Problema 3. Resolver la ecuación $2x^2 + bx + c = 0$, donde b y c son parámetros, es decir, pueden adquirir cualquier valor.

Algoritmo de solución

1. Introducir los valores de los parámetros b y c.
2. Calcular $D = b^2 - 4 \times 2 \times c$.
3. Si $D \geqslant 0$ pasar al punto 4; si $D < 0$ pasar al punto 9.
4. Extraer la raíz de D.

5. Calcular $x_1 = \frac{-b+\sqrt{D}}{2\times 2}$.

6. Calcular $x_2 = \frac{-b-\sqrt{D}}{2\times 2}$.

7. Escribir la respuesta.

8. Terminar el trabajo.

9. Dar el aviso: «La ecuación no tiene soluciones».

10. Terminar el trabajo.

Todo algoritmo que se emplea en el ordenador debe poseer cinco propiedades fundamentales: debe ser finito, preciso, eficaz, debe tener sus datos de entrada (iniciales) y sus datos de salida (resultado).

Examinemos más detalladamente cada una de estas propiedades.

1) Se dice que el algoritmo es finito si conduce a la solución del problema en un número finito de operaciones cualesquiera que sean los datos iniciales.

En los problemas NN 1, 2, 3 que examinamos los algoritmos son finitos ya que, independientemente de los datos de entrada, la solución consta de no más de 10 operaciones.

Expongamos un ejemplo de algoritmo que no es finito.

Problema 4. Obtener la sucesión de números naturales, comenzando desde el número N.

Algoritmo de solución

1. Tomar el número natural N.

2. Aumentar su valor en 1.

3. Escribir la respuesta (el valor del número natural inmediato).

4. Pasar al punto 2.

Este algoritmo conduce a un número infinito de operaciones. (Más tarde veremos que el ordenador, cumpliendo el algoritmo dado, interrumpirá, sin embargo, el cálculo, pues obtendrá un número tan grande que no cabrá en la célula de la memoria).

2) Precisión: es la exigencia de que cada paso del algoritmo quede determinado unívocamente.

Los algoritmos de solución de los problemas NN 2, 3, 4 poseen la propiedad de precisión. Aquí, en todas las condiciones, está determinado unívocamente cada paso. Pero en el algoritmo de solución del problema N° 1 no hay precisión, ya que no se señalan las operaciones para el caso en que el tren no está en la estación.

3) Eficacia significa que el algoritmo debe resolver con suficiente sencillez el problema planteado.

Por ejemplo, de Dubná a Moscú se puede ir según el algoritmo N° 1, pero también se puede así:

1. Venir a la estación «Dubná».

2. Comprar un billete hasta la estación siguiente.

3. Sentarse en el tren.

4. Bajar del tren en la estación siguiente.

5. Si no es Moscú pasar al punto 2; si la estación es Moscú pasar al punto 6.

6. Terminar el viaje.

Este algoritmo no posee la propiedad de eficacia, pero resuelve correctamente el problema planteado.

4) El algoritmo debe tener datos iniciales.

En el problema N° 1 éstos son la existencia del tren y de dinero para comprar el billete.

En los problemas N°N° 2, 3 son los valores de los coeficientes *a*, *b* y *c*.

5) Todo algoritmo debe tener datos de salida, es decir, resultado.

En el problema N° 1 este resultado es la llegada a Moscú, y en los problemas N°N°2, 3 los resultados son los valores de las raíces de la ecuación.

Ejercicio 1. Elaborar los algoritmos de solución de los problemas siguientes:

Problema 6. Cocer fideos.

Problema 7. Resolver la ecuación $ax = 2$ (a puede adquirir cualquier valor).

Problema 8. Resolver la desigualdad $ax > 2$.

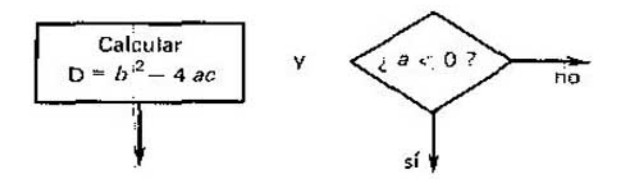
1.5. Esquema en bloque

Es más cómodo describir el algoritmo con esquemas en bloque que con palabras, es decir, en forma de ciertas figuras geométricas unidas por flechas. Estas flechas mues-

tran los enlaces lógicos entre los distintos pasos de solución del problema: sucesión de ejecución, condiciones en las que se realizan unas u otras operaciones.

Existe un convenio según el cual se eligen determinadas figuras geométricas para designar el tipo del paso. Designemos el bloque de control de la condición con un rombo, y todos los bloques restantes con rectángulos.

Ejemplo.

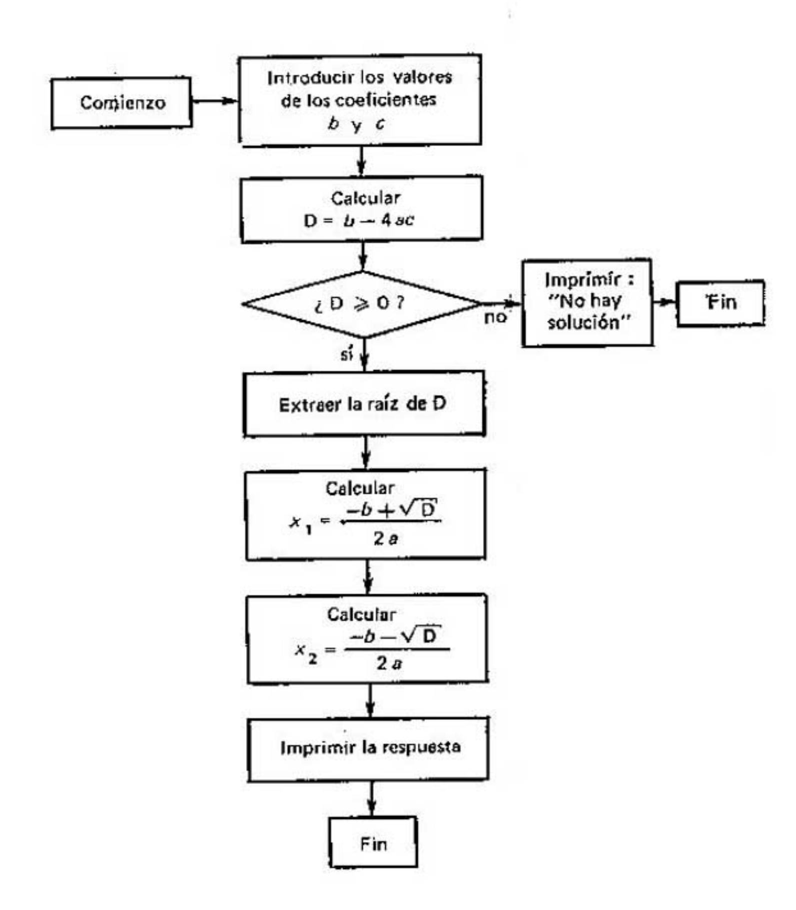


Desde el rectángulo siempre sale una sola flecha; desde el rombo salen no menos de dos. Esto significa que la solución debe transcurrir por una de las distintas vías posibles en dependencia del cumplimiento de las condiciones, pero para los datos concretos iniciales la solución transcurre solamente por una vía.

Describiremos con ayuda del esquema en bloque el algorítmo de solución del problema:

Resolver la ecuación $2x^2 + bx + c = 0$.

Esquema en bloque del algoritmo

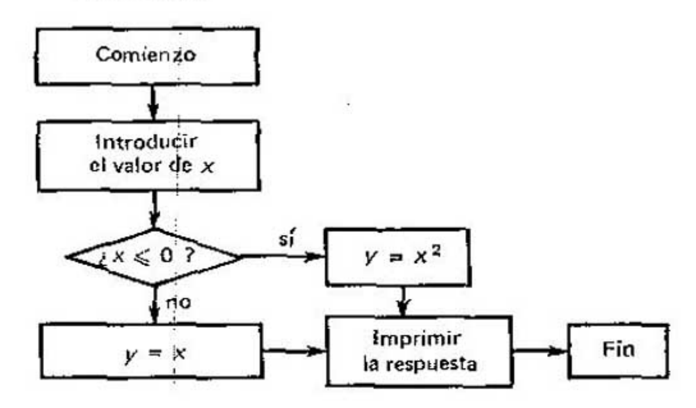


Ejercicio 2. Escribir esquemas en bloque para los problemas 1, 2, 4, 5, 8.

Problema 9. Elaborar el esquema en bloque del cálculo

$$y(x) = \begin{cases} x^2, \text{ si } x \leqslant 0; \\ x, \text{ si } x > 0. \end{cases}$$

Solución.



Ejercicio 3. Elaborar el esquema en bloque del cálculo

$$y(x) = \begin{cases} \operatorname{sen} x, \text{ si } x \leqslant -2; \\ \cos x, \text{ si } x > -2. \end{cases}$$

Ejercicio 4. Elaborar el esquema en bloque del cálculo de las funciones siguientes:

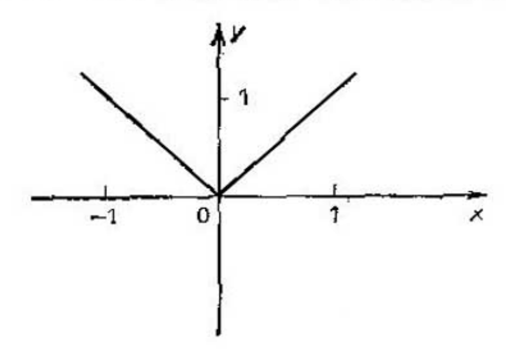
Problema 10.

$$y(x) = \begin{cases} 1/x, \text{ si } x \neq 0; \\ 0, \quad \text{si } x = 0. \end{cases}$$

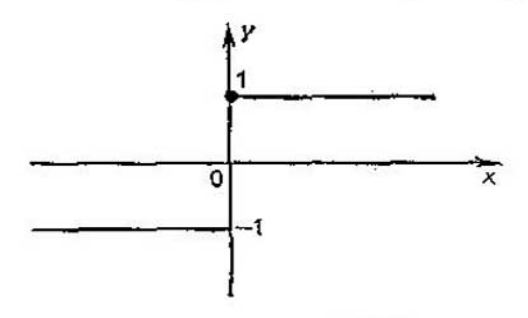
Problema 10a.

$$y(x) = \begin{cases} \operatorname{sen} x, \text{ si } \operatorname{sen} x < 0; \\ 1, \qquad \text{si } \operatorname{sen} x = 0; \\ \dfrac{\operatorname{sen} x}{x}, \text{ si } \operatorname{sen} x > 0. \end{cases}$$

Problema 11. $y(x)$ está dada por el gráfico



Problema 12. $y(x)$ está dada por el gráfico



Problema 13. $y(x) = \sqrt{x - 7}$.

Para la expresión subradical negativa dar el aviso: «no hay solución».

Problema 14.

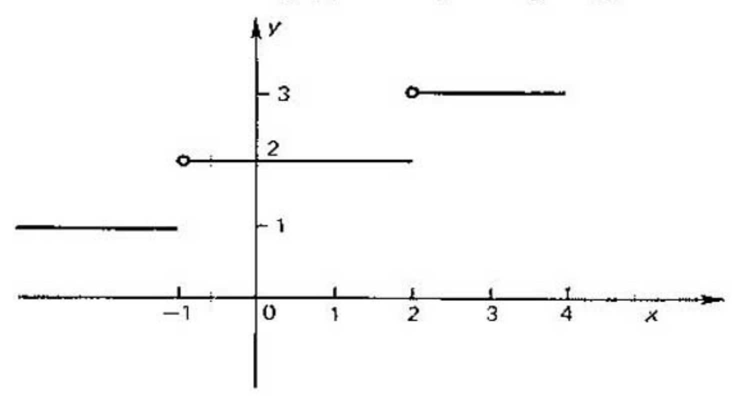
$$y(x) = \begin{cases} x - 1, & \text{si } x \leqslant 1; \\ x^2 - 1, & \text{si } 2 > x > 1; \\ x - 2, & \text{si } x \geqslant 2. \end{cases}$$

Nota. En un bloque no se permite comprobar la desigualdad doble: $2 > x > 1$.

Problema 15.

$$y(x) = \begin{cases} \text{sen}\, x, & \text{si } x \leqslant -4; \\ \cos x, & \text{si } -2 > x > -4; \\ \text{tg}\, x, & \text{si } -1 > x \geqslant -2; \\ x^3, & \text{si } x \geqslant -1. \end{cases}$$

Problema 16. $y(x)$ se da por el gráfico



Problema 17.

$$y(x) = \frac{1}{x^2 - 3x + 2}.$$

Problema 18.

$$y(x) = \frac{1}{\sqrt{x^2 - 3x + a}},$$

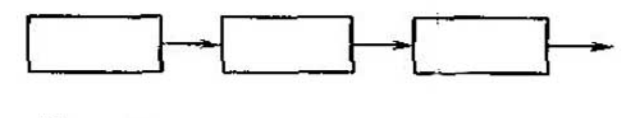
aquí a es un parámetro, es decir, puede tener cualquier valor.

1.6. Tipos fundamentales de algoritmos

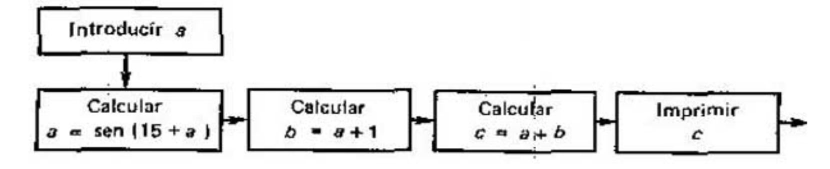
Existen algoritmos lineales, cíclicos y algoritmos con ramificaciones.

Los algoritmos en los que cada operación siempre se ejecuta, además, una sola vez, son *lineales*.

El esquema en bloque del algoritmo lineal representa en sí una cadenita de rectángulos a cada uno de los cuales conduce una sola flecha.



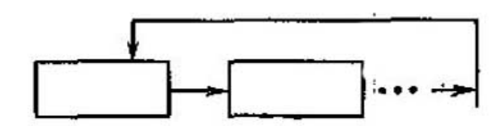
Ejemplo.



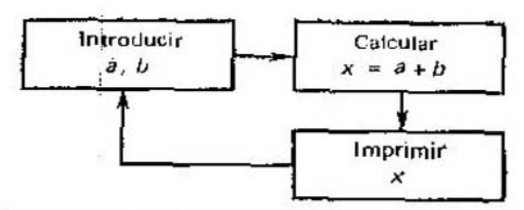
Ejercicio 5. Indiquen cuáles de los algoritmos de soluciones de los problemas 1—18 son lineales.

Son *cíclicos* aquellos algoritmos en los que unas mismas operaciones se repiten reiteradamente. Las mismas operaciones que se repiten forman un ciclo.

En los esquemas en bloque de semejantes algoritmos existe obligatoriamente una línea cerrada de flechas.



Ejemplo.



Ejercicio 6. Indique cuáles de los algoritmos de las soluciones de los problemas 1—18 son cíclicos.

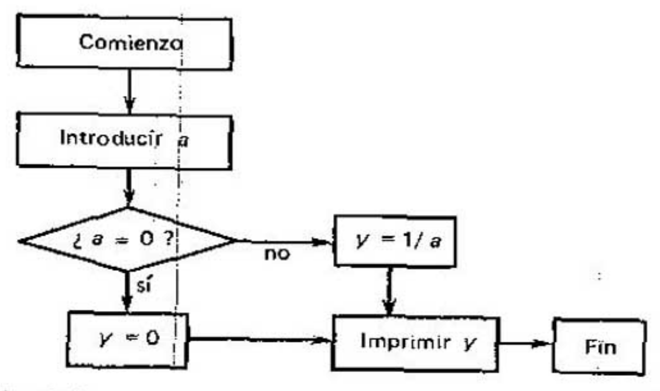
El alogoritmo que contiene cualquier condición es un algoritmo *con ramificaciones.*

En el esquema en bloque de semejante algoritmo hay obligatoriamente, por lo menos, un bloque (bloque condición) desde el que salen dos o tres flechas.

Ejemplo.

Supongamos que hay que calcular

$$y = \begin{cases} -\frac{1}{a}, \text{ si } a \neq 0; \\ 0, \text{ si } a = 0. \end{cases}$$



El bloque de condición determina siempre la única dirección para continuar la solución. Si los valores iniciales de los datos son otros esta dirección puede ser diferente, pero siempre la única. Por esto, en los algoritmos con ramificaciones una parte de bloques no trabaja con unos datos iniciales, mientras que otra parte puede no trabajar con otros datos iniciales.

Así, en el ejemplo expuesto, si $a = 0$ entonces el bloque $\boxed{y = \frac{1}{a}}$ no trabajará en absoluto.

Ejercicio 7. Determinar cuáles de los algoritmos de solución de los problemas 1—18 son algoritmos con ramificaciones.

Ejercicio 8. Elaborar el esquema en bloque de solución:

a) de la ecuación cuadrada $ax^2 + bx + c = 0$, donde a, b, c son parámetros (pueden adquirir cualquier valor, incluyendo el nulo).

b) $ax = b$;

c) $\sqrt{1 - \operatorname{sen} A} \times (x + 1) = 1$;

d) $\sqrt{2 - a} \times (x - 380) = 0$;

e) $ax > b$.

Ejercicio 9. Reducir el valor de un ángulo arbitrario α a un valor que yazca en $[0, 2\pi]$.

Al escribir un algoritmo detallado de solución de cualquier problema complejo se puede librar a otras personas de la necesidad de romper la cabeza en la búsqueda de procedimientos de su solución. En el mun-

do se han acumulado bibliotecas enteras de algoritmos de solución de los problemas de la ciencia y de la técnica que con mayor frecuencia se encuentran.

Una vez propuesto el algoritmo, se puede encargar la solución del problema a otra persona, que solamente debe seguir mecánicamente el algoritmo. El siguiente paso en esta dirección es el deseo de encargar este trabajo al ordenador que debe «comprender» el algoritmo y seguirlo puntualmente.

1.7. Noción de los lenguajes de programación

Ya hemos estudiado dos procedimientos de describir el algoritmo: en lengua española ordinaria y con ayuda de los esquemas en bloque. Estas descripciones son comprensibles para el hombre, pero no para el ordenador. El programa es la descripción del algoritmo que pueden comprenderla tanto el hombre como el ordenador.

En los primeros años del desarrollo de la programación el hombre escribía el algoritmo de solución de los problemas en forma de una secuencia de instrucciones de máquinas. Una instrucción es un código numérico que contiene información respecto a cuál operación y sobre cuáles números debe realizar la máquina.

Para cada tipo de ordenador existe su propia tabla de códigos para designar las diferentes operaciones de la máquina. Pre-

cisamente el algoritmo se escribe totalmente con estas designaciones. El programa que se compone de instrucciones de máquina se denomina programa en códigos.

La programación en códigos tiene una serie de defectos: bajo rendimiento del programador, mucho tiempo y fuerzas en la escritura del algoritmo en los códigos de la máquina, un trabajo agotador, frecuentes errores, cuya búsqueda también es muy difícil. Además, los programas compuestos para un ordenador no sirven absolutamente para una máquina de otro tipo.

La creación de lenguajes especiales de programación fue un acontecimiento importante. En estos lenguajes se describen los algoritmos de solución de problemas, y los ordenadores pueden «comprender» y ejecutar estos algoritmos.

Los lenguajes de programación (o lenguajes algorítmicos) permiten las designaciones habituales: « + », « — » y etc., la utilización de letras y ciertas palabras. Además no es necesario transformar manualmente semejante programa en códigos de máquina: ésta lo hace por sí misma, con ayuda del programa de traducción (compilador).

El compilador es un programa especial, que está escrito previamente por programadores de alta calificación y que se almacena en la memoria del ordenador. El volumen del programa de traducción es de miles (a veces decenas de miles) de instrucciones de máquina.

1.8. El lenguaje Fortran

1.8.1. Alfabeto del lenguaje Fortran

Uno de los lenguajes de amplia divulgación en el mundo es el Fortran, creado en los EE.UU. en 1956. El lenguaje Fortran adoptó este nombre de las palabras inglesas *formula translation*.

Como cualquier lenguaje el Fortran tiene su alfabeto, es decir, un juego de símbolos con los que se componen palabras y oraciones.

El alfabeto del Fortran lo componen:

— las letras latinas mayúsculas desde A hasta Z;

— las cifras árabes desde 0 hasta 9;

— los signos + — */ , = ( ) ′.

Conjuntamente con los símbolos señalados en el programa se pueden utilizar los blancos: espacios vacíos entre las letras, cifras, signos.

En aquellos casos cuando sea necesario señalar exactamente cuántos blancos se deben poner entre otros símbolos designaremos al blanco como ␣.

1.8.2. Constantes numéricas del lenguaje Fortran

Las constantes del Fortran son números que se utilizan en las diversas operaciones. Cada constante que se encuentra en el programa se aloja en su célula de la memoria y se encuentra allí en el transcurso de todo

el funcionamiento del programa dado. Estudiaremos las constantes de dos tipos: enteras y reales.

Las constantes reales son fracciones decimales, así como números enteros escritos en forma de fracciones decimales.

Cada fracción decimal contiene una coma que separa su parte entera de la fraccional. En el lenguaje Fortran en lugar de la coma se utiliza el punto.

Ejemplo. Al número 3,14 le corresponde la constante del Fortran 3.14.

En cada constante real de este tipo debe estar presente el punto. Si el número no contiene parte entera, o parte fraccional, se puede no escribir los ceros.

Ejemplo. El número 0,18 se puede escribir en forma de .18 o bien 0.18; el número 35 se escribe en forma de 35. o bien 35.0.

En la ciencia y en la técnica, para los números muy grandes o muy pequeños, se utiliza la escritura mediante la potencia de diez.

Por ejemplo, $2{,}5 \times 10^9$; $1{,}8 \times 10^{-12}$.

En el Fortran se prevé el tipo correspondiente de constantes.

Al número $2{,}5 \times 10^9$ le corresponde la constante 2.5E9, y al número $1{,}8 \times 10^{-12}$ le corresponde la constante 1.8E−12.

Como se ve de los ejemplos la letra E en la escritura de las constantes significa lo siguiente: «multiplicar por diez a la potencia». El aspecto general de la constante de este tipo es m. kEn. Aquí m. k es la mantisa, mientras que En es la potencia de 10 tipo 10^n.

En el Fortran un mismo número puede representarse de distintas maneras.

Ejemplo. Al número 5,5 le corresponden las escrituras: 5.5; 5.5E1; 55.E−1.

El signo de la propia constante se pone delante de ella, y el signo del exponente de diez se pone después de la letra E (se puede no escrbir el signo +).

Nota. La constante de tipo 10^n se debe escribir como 1. En y no como En. Por ejemplo 1.E5, y no E5.

Las constantes enteras tienen el aspecto de números enteros ordinarios.

Por ejemplo, la constante entera 15 corresponde al número entero 15 (sin punto).

En el Fortran un mismo número puede ser escrito bien en forma de constante real (con punto) bien en forma de constante entera (sin punto). En ciertos casos, como verán después, en principio, es importante el poner o no poner este punto.

Las constantes entera y real que corresponden a un mismo número se diferencian entre sí en la máquina por su aspecto y por las reglas de operación con ellas.

1.8.3. Diapasón de variación de las constantes

En el ordenador electrónico las constantes pueden variar en un diapasón determinado. Un número demasiado grande no cabe en la célula del ordenador, mientras que el

demasiado pequeño la máquina lo considera como cero («cero de máquina»).

En el ordenador EC las constantes reales están comprendidas, por su módulo, entre los números 10^{-78} y 10^{76}; el cero de máquina es cualquier número que por su módulo sea inferior que el número 10^{-78}. Las constantes enteras no superan en la EC ЭВМ*) 2×10^9.

Ejercicio 10. Escribir en Fortran los números: 78,05; 0,0000002; —3800000; 18,905; $3,5 \times 10^{-10}$; $1,873 \times 10^{15}$; 0,000000000729; —123000000; —0,000000123; 10^7; -10^5; -10^{-13}.

Ejercicio 11. ¿A qué números corresponden las siguientes constantes del Fortran

1.5E—4; —120; —1.20; 3.; .3; .1E4?

Ejercicio 12. 1. Escribir en Fortran los números: $+3,8 \times 10^5$; $13,7 \times 10^{-10}$; 10^3; —0,0001; —150/7,5; 3^5; -10^8;

2. Escribir a cuáles números les corresponden las siguientes constantes del Fortran:

—3.87; 3.E—7; 1.E1; .2E2; —.012E—4; 1.42E—1

1.8.4. Magnitudes variables simples

Como se dijo anteriormente, el programa compilador aloja las constantes en las células de la memoria, y estas células no cambian su contenido en el transcurso de todo el funcionamiento del programa. Las magnitudes variables se almacenan también en las células de la memoria, pero en éstas se inscriben los valores corrientes de

---

* EC ЭВМ = Sistema Unificado de Ordenadores. (*N. de la R.*)

las variables, por lo que el contenido de dichas células varía durante el tiempo de funcionamiento del programa.

Al calcular el nuevo valor de la variable tiene lugar la sustitución del contenido anterior de la célula de la memoria por otro valor nuevo.

Toda variable debe ser designada. Generalmente, en la literatura científica se ha adoptado designar las magnitudes variables con una letra, por ejemplo,

$$H, t, R^*, B^1, \alpha, B_1, V_0, V_t.$$

Con esto se utilizan letras latinas, rusas, griegas, tanto mayúsculas como minúsculas. Además, las letras frecuentemente se acompañan de distintos signos.

En el Fortran solamente se emplean letras latinas mayúsculas, que en total son 26. Por esto, para la designación de las variables se utilizan las combinaciones de letras y cifras. Semejantes combinaciones se denominan nombres de las variables. En lugar del término «nombre» se utilizan también los términos «denominación», «identificador».

El nombre de la variable lo elige el programador en concordancia con las reglas siguientes:

El nombre debe comenzar por una letra, estar compuesto solamente por letras latinas y cifras árabes y debe contener no más de seis símbolos.

Ejemplo. A, B1, GAMMA, DUBNA, MOSCOW.

Ejercicio 13. Señalen cúales de las combinaciones de símbolos expuestas pueden ser nombres de variables:

Г5; D, DIV, C3A7, 3CA7, MOSCOW, DUBNA, β, BETA.

Las variables, igual que las constantes, pueden ser de tipo «entero» y de tipo «real».

La variable entera solamente puede tomar valores enteros; la variable real solamente puede tomar valores reales.

Lo mismo que el tipo de constante se puede determinar por su aspecto exterior, el tipo de variable también puede determinarse por su nombre.

En el Fortran se ha adoptado la regla siguiente. Si el nombre de la variable comienza por una de las letras I, J, K, L, M, N, entonces la variable es de tipo «entero», y si el nombre comienza por cualquier otra letra entonces la variable tiene el tipo «real».

Existen también otros procedimientos de determinación del tipo de la variable, pero no vamos a estudiarlos.

Ejemplo. Las variables K1, L3410, MASTER, MOSCOW son de tipo entero, mientras que las variables AK1, BL3410, FMAST, DUBNA, ALFA, TETA, son de tipo real.

Ejercicio 14. Distribuyan por tipos las variables:

KPA, AKP, DELTA, LOTO, A11, LA11, H15, GAMMA, MAG, ALOG

Ejercicio 15. Designen 5 variables enteras y 5 variables reales.

1.8.5. *Expresiones aritméticas*

Así pues, hemos conocido las constantes y variables del Fortran. Ahora pasaremos a la escritura de las operaciones con ellas. En el Fortran se utilizan los siguientes signos de operaciones aritméticas:

+ adición,

— sustracción,

* multiplicación,

/ división,

** elevación a potencia.

Ejemplo. La expresión a^2 en el Fortran se escribe A**2;

$(ab)^3$, como (A*B)**3;

$\frac{15a}{3}$, como 15*A/3;

$\frac{a+b}{c+d}$, como (A+B)/(C+D)

La primacia de las operaciones aritméticas es la siguiente:

a) ** (es la operación primacial),

b) * y /,

c) + y —.

El orden de operaciones durante los cálculos coincide con el adoptado en la matemática, pero las operaciones de una misma primacia se ejecutan estrictamente en el orden de su escritura.

Por ejemplo, al encontrar la expresión 15*R/3 la máquina primero multiplica por R y después divide por 3. El hombre pri-

mero dividirá 15 por 3 y luego multiplicará por R.

La expresión R* (15/3) la máquina la calculará de otro modo: primero ejecutará la operación entre paréntesis y luego multiplicará por R.

Ejercicio 16. Escribir en Fortran:

$$\frac{ab}{cd}; \quad \frac{ab}{c}\times d; \quad \frac{a}{c}\times\frac{b}{d}; \quad \frac{a^3}{b^3}$$

Ejercicio 17. Poner el orden de ejecución de las operaciones por la máquina:

A*B/(C + D**3)*F/E

En el programa también se pueden utilizar funciones matemáticas elementales las que estudiaremos posteriormente con más detalle.

He aquí algunas de éstas:

Función	Escritura en Fortran
$\lvert x \rvert$	ABS(X)
$\sqrt{x}$	SQRT(X)
sen x	SIN(X)
cos x	COS(X)
ln x	ALOG(X)

El argumento de la función siempre se pone entre paréntesis.

Ejemplo. La expresión a^2 sen ωt en Fortran se escribe

A**2*SIN(OMEGA*T)

Ejercicio 18. Programar las expresiones:

$a+\sqrt{b}$; $\frac{a-2b}{2c}$; $\text{sen}^2 x - \cos^2 x$; $\log^3 x$

Las constantes numéricas, las variables, las funciones, las combinaciones de constantes, variables y funciones, separadas por signos de operaciones aritméticas y paréntesis ordinarios, forman las expresiones aritméticas.

Ejemplo. A; 5; 5*A; (A*5)**2; SIN (X)*COS (B); SQRT (C); SIN (X**2); SIN (X)**2

Las expresiones aritméticas corresponden a las expresiones matemáticas habituales, escritas según las reglas del Fortran.

Ejercicio 19. Restituir el aspecto universalmente admitido de las expresiones del ejemplo anterior.

Al escribir expresiones aritméticas se debe tener en cuenta lo siguiente:

a) no se pueden poner consecutivamente dos operaciones aritméticas, Por ejemplo, la escritura 2*—3 es inadmisible, y se debe escribir

2*(—3);

b) no se puede omitir el signo de multiplicación. Por ejemplo, el producto $2a$ se debe escribir como 2*A;

c) cualquier operación aritmética con dos constantes enteras o con dos variables enteras da un resultado entero.

Ejercicio 20. Son admisibles o no las siguientes expresiones aritméticas:

a) —B—SQRT(B*2—4AC);

b) (—B+SQRT(B**2—4*A*C))/2*A

Nos detendremos más detalladamente en la operación de división de constantes o variables enteras.

En el Fortran se ha aceptado la siguiente regla de obtención del resultado: al dividir unas por otras constantes o variables enteras los signos decimales del cociente se omiten.

Ejemplo.

Al dividir 4 por 5 el resultado es 0
Al dividir 39 por 10 el resultado es 3
Al dividir —9 por 2 el resultado es —4
Al dividir 9999 por 1000 el resultado es 9

Subrayaremos que durante la división de enteros el cociente no se redondea.

Durante el trabajo con constantes enteras y variables enteras se puede obtener en el ordenador un resultado inesperado que refuta, por ejemplo, la afirmación de que «dos por dos son cuatro». Ilustraremos esto en el ejemplo siguiente.

Examinemos la expresión:

(2/3*3)*2

Entre paréntesis está escrita la expresión $\frac{2*3}{3}$ que, como resultado, debe dar 2; después de multiplicar 2 por 2, como es sabido, debe obtenerse 4. Pero en el ordenador electrónico al dividir 2/3 se obtiene cero (se omite la parte fraccionaria). Después de multiplicar este cero por 3 y por 2 como resultado se obtendrá cero ¡y no cuatro!

Ejercicio 21. Programe la expresión $\frac{15\times 3}{4}$ con dos procedimientos diferentes:

a) tal que primero se ejecute la operación de multiplicar;

b) tal que primero se ejecute la operación de dividir.

Comparen los resultados.

El tipo del resultado de la expresión aritmética se determina por el tipo de las variables, constantes y funciones que entran en esta expresión.

El resultado del cálculo de una expresión aritmética tiene tipo de «entero» entonces, y solamente entonces, cuando en esta expresión entran sólo constantes enteras y variables enteras.

Ejemplo. El resultado de la expresión (3*1/8**2)/100 tiene el tipo de «entero» mientras que el resultado de (3.*1/8**2)100 tiene el tipo de «real» ya que una de las constantes de la segunda expresión (3.) tiene el tipo «real».

Ejercicio 22. Escribir el resultado del cálculo de las expresiones:

a) 80/6; b) 80/6.; c) 5/6*6; d) 5./6*6;
e) 5/6*6.; f) 6/7; g) —5/4; h) 4/2*2;
i) 5/2*2; j) 5./2*2

Ejercicio 23.

1. Programar las expresiones:

a) $\frac{-b+\sqrt{b^2-4ac}}{2a}$; b) $\frac{\operatorname{sen} x}{\cos^2 x+1}$;

c) $\operatorname{sen}\left(\frac{a+b}{c-d}\right)$;

2) Restituir las fórmulas:
a) ALOG (SIN(X)) —
b) A*B/C*K; c) A—SQRT(A*(A+B))/D
3. Escribir el resultado:
a) 25/4*4—25 b) 25./4*4—25 d) 24/4*4—24
d) 24/4.*4—24 e) 25/4*4.—24 f) 25/4*4—25
g) 24/4*4.—24

1.9. Operadores del lenguaje Fortran

El programa de la solución de cualquier problema en el ordenador es la secuencia de instrucciones concretas a la máquina: introducir datos, calcular el valor de una expresión, imprimir el resultado, etc. Cada indicación aislada del programa Fortran se denomina operador. El operador, como regla, es una palabra inglesa, acompañada de información adicional en forma de variables y constantes. La palabra inglesa indica la operación que se debe realizar (READ significa leer, PRINT es imprimir, RETURN es regresar, etc.)

En el Fortran se utilizan cerca de veinte palabras inglesas. Si el programador permite cometer errores en estas palabras, entonces el programa traductor, denominado también compilador, que compara las palabras con los patrones, dará a la impresora una información respecto al error (diagnosis).

Título del programa. El primer operador de cada programa debe ser el operador

PROGRAM name

donde *name* es el nombre del programa. Este nombre se elige arbitrariamente, pero teniendo en cuenta las reglas de la escritura del nombre en el Fortran: solamente debe contener letras y cifras, comenzar por una letra y estar compuesto por no más de 6 símbolos. Es cómodo dar al programa un nombre que esté relacionado con la denominación del problema a resolver. Por ejemplo, si se programa la composición de cualquier tabla el programa se puede llamar TAB; si se calcula el sen x, entonces el programa se puede llamar SINUS, etc. En los programas para la EC ЭВМ el operador PROGRAM no existe.

El operador aritmético de atribución tiene el aspecto

X=A

donde X es el nombre de la variable, A, la expresión aritmética.

En el ordenador el operador de atribución se efectúa así: se calcula el valor de la expresión aritmética y el resultado se inscribe en la célula que corresponde a la variable X. El signo «=», que por su escritura coincide con el signo de igualdad, significa el envío del valor numérico a la célula de la memoria.

Todas las variables que entran en la expresión aritmética A deben ser determinadas hacia el momento de la ejecución del operador dado.

Ejemplo. B = 2.—3.5
X5 = B + 2.**5
CAP = 3./B + X5

Aquí los valores de las expresiones aritméticas tienen tipo «real» y se atribuyen a las variables reales. No obstante, el tipo de la variable también puede no coincidir con el tipo de la expresión aritmética (por ejemplo, X = 5—3). ¿Qué hacer en este caso? En el Fortran rige la regla: *a la célula se envía el número en forma que corresponda al tipo de aquella variable que está en la parte izquierda del operador de atribución.*

Por ejemplo, después de ejecutar el operador X = 5 — 3 en la célula X estará el número 2. (es decir, el 2 «real»). Si se ejecuta el operador K = 5./7., entonces el resultado debe ser transformado al tipo de «entero», es decir, en el resultado se omite la parte fraccionaria. Así, pues, una vez ejecutado el operador K = 5./7. en la célula K se inscribirá el cero «entero».

Los operadores del tipo A = A + 1 desconciertan frecuentemente a los programadores principiantes. Sin embargo, en este operador no hay nada extraño: al número que se encuentra en la celda A se le añade 1., y el resultado se inscribe en la misma celda.

Ejercicio 24. Escriba en forma de operadores de atribución las fórmulas siguientes:

a) $x = \frac{b^2 + c^2}{d}$, b) $E = \frac{35x + y}{a^3}$, c) $A = \frac{3c}{25d}$.

2. ¿A qué fórmulas corresponden las siguientes expresiones aritméticas: a) A/B*(C+D); b) A*(C+D)/B; c) A/B/(C+D); d) A/B*C+D?

Ejercicio 25. ¿Qué valores se atribuirán a las variables K y A?

a) K=1./2.; b) K=3./2.+1.2; c) K=3/2+1/2; d) 1*7/8; e) A=9/8*7.

Operador de salto indudable u operador imperativo GO TO. En el programa los operadores del Fortran se escriben uno debajo de otro, cada uno de ellos en un renglón nuevo. Estos operadores se ejecutan por la máquina en el orden de escritura, hasta que se encuentre un operador que indique cualquiera otro orden de su ejecución. Uno de estos operadores es

GO TO n

donde n es el número añadido a la izquierda a aquel operador a cuya ejecución debe pasar el ordenador electrónico. Este número n se denomina *marca* (*etiqueta*) *del operador.*

El valor n debe encontrarse en los límites de 1 hasta 99999 para la EC ЭВМ. El programador puede elegir las marcas según su deseo, pero se debe cuidar de que en los límites de un mismo programa no hayan marcas iguales. Es cómodo componer el programa de tal manera que desde el comienzo hasta el final de éste las marcas (números) crezcan monótonamente.

Ejemplo. Atribuyamos al operador A = 15.3 la marca 10. 10␣A = 15.3

Si se necesita, ejecutar precisamente

éste, «saltando» varios operadores del programa, entonces escribiremos en el programa el operador GO TO 10.

El paso a la ejecución del operador con la marca n frecuentemente se denomina *transferencia o salto del mando al operador con la marca n.*

Ejemplo. Supongamos que se ha dado la sucesión de operadores

```
    B = 15.3
    D = 8.6
    GO TO 123
100 B = — 1.2
    D = 2.2
123 A = B + D
```

Después de ejecutar los primeros tres operadores el mando se transfiere al operador con la marca 123, por lo que en la célula A se alojará el número 23.9 = 15.3 + + 8.6.

Ejercicio 26. Escriba un operador delante de todos los operadores del ejemplo anterior para que en la célula A se obtenga el resultado 1.

Operadores END y PRINT. El último operador de todo programa debe ser el operador END. Haciendo uso del operador PROGRAM y los operadores de atribución y END, ya se pueden confeccionar programas. Por ejemplo, escribamos el programa de cálculo del discriminante de la ecuación cuadrática $1{,}32x^2 + 2{,}75x - 3{,}45 = 0$.

```
PROGRAM DISKR
A = 1.32
B = 2.75
C = — 3.45
D = B**2 — 4. *A*C
END
```

Semejante programa no tiene interés alguno, ya que el resultado D, calculado por la máquina, no será comunicado al programador.

El operador

PRINT n, D

sirve para extraer la información del ordenador a la impresora.

Aquí D es una variable cuyo valor debe ser imprimido, y n es la marca del operador FORMAT, en combinación con el cual trabaja PRINT.

En el operador PRINT se indican los valores de las variables que deben ser imprimidas, y en el operador FORMAT existe la información respecto a cómo situar estos números en el papel, la cantidad de signos que se deben conservar en el número, el texto que debe acompañar los números extraídos. Por ejemplo, el operador FORMAT se puede escribir de tal manera que el valor calculado del discriminante D se imprimirá así:

EL DISCRIMINANTE DE LA ECUACION 1.32*X**2 + 2.75*X — 3.45 = 0

HAY D = 25.78

El programador debe reflexionar con anticipación sobre la forma en la que quiere obtener el resultado. Es útil saber que una página de la impresora ancha puede contener hasta 66 renglones con 144 símbolos en cada uno de ellos (es mejor utilizar no más de 120).

A cada puesto del símbolo en el papel le corresponde una posición del dispositivo impresor. Por esto se dice que el texto que se compone de cinco símbolos ocupa 5 posiciones, la constante entera de tres signos ocupa tres posiciones, pero la real de tres signos ocupa cuatro posiciones, pues el punto que contiene la inscripción del número real (283., 5,14, etc.) es un símbolo equitativo. Al imprimir no se recomienda ocupar la primera posición, pues pueden surgir regímenes de trabajo del dispositivo imprimidor no planificados.

Operador FORMAT. Su aspecto general es:

n FORMAT (lista de especificaciones)

Aquí n es la marca, y entre paréntesis está comprendida la información respecto a cómo imprimir y distribuir en el papel los resultados que se dan.

Si el resultado tiene tipo de «entero» entonces se utiliza la *especificación* Im. Aquí I es el índice del tipo «entero», y m es el número de posiciones en el papel, que se asignan para imprimir el resultado (incluyendo la posición para el signo menos, si éste existe). Además, la *cifra de rango inferior* del resultado ocupa siempre la posi-

ción más a la derecha de todas las m posiciones. Si para el resultado se necesitan menos posiciones que m, entonces las posiciones de la izquierda, que superen las requeridas, serán blancos. Por ejemplo, si el valor de la variable que se deduce es igual a —150, entonces en el operador FORMAT es suficiente señalar la especificación I4. Si se elige, digamos, la especificación I7, entonces en el papel, a la izquierda de —150, habrá 3 blancos. Si, por el contrario, se señalan menos posiciones que se necesitan, entonces solamente se imprimirán las cifras de rangos inferiores del resultado.

Si el resultado tiene tipo «real» entonces se utiliza la *especificación* Fm.n. Aquí F es el índice tipo «real», m es el número total de posiciones en el papel designadas para la impresión del resultado, y n, el número de posiciones designadas para la parte fraccionaria.

Por ejemplo, para imprimir el resultado —15,347 se puede utilizar la especificación F7.3. Aquí 7 es el número total de posiciones designadas para el alojamiento del resultado (incluyendo las posiciones para los signos menos y punto), 3, el número de posiciones para la parte fraccionaria del resultado. Si se imprime —15,347 por la especificación F10.3 entonces 3 posiciones de la izquierda estarán vacías. Si se señala m menor de la que se necesita para colocar el resultado dado, entonces en la impresión situada más a la izquierda de todas las m posiciones se representará el símbolo *.

Por ejemplo, al extraer el número 3.14 por la especificación F3.2 en la impresión obtendremos *14.

Los valores de distintas variables reales se pueden imprimir por especificaciones iguales, eligiendo las correspondientes m y n.

Ejercicio 27. Escribir las especificaciones de extracción de los números 9,8; +2,128; 103,14; 15; —728; —25,5.

Al extraer por las especificaciones Im y Fm.n se debe saber el orden aproximado de los números que salen, para elegir correctamente m y n. Aquí m es el número total de posiciones designadas para todo el resultado, y n es lo mismo, pero para su parte fraccionaria. Con esta especificación se extrae n ó n + 1 cifras significativas (en dependencia del tipo del ordenador electrónico). El valor de m se elige de tal modo que $m \geqslant n + 7$.

Si se necesita imprimir valores de varias variables entonces sus nombres se enumeran en el operador

PRINT (lista de las variables)

Por ejemplo, 4 variables A,B,C,D se extraen a la impresora por el operador

PRINT␣5, A, B, C, D.

En el operador FORMAT que corresponde a este PRINT, se señalan 4 especificaciones de salida (una para cada variable). Por ejemplo,

PRINT 5, A, B, C, D
5 FORMAT (E8.3, F12.7, F5.1, F9.4)

Frecuentemente, para la extracción de varias variables se utiliza una misma especificación. Por ejemplo, las variables A, B, C, D se pueden extraer por la especificación F12.7:

PRINT 5, A, B, C, D
5 FORMAT (F12.7, F12.7, F12.7, F12.7)

Aquí F12.7 se repite 4 veces. Para no escribir repetidamente una misma especificación se puede, delante de ésta, escribir el coeficiente de repetición:

PRINT 5, A, B, C, D
5 FORMAT (4F12.7)

Si un grupo entero de especificaciones se repite varias veces entonces el coeficiente de repetición se pone delante de los paréntesis, entre los que se encierra el grupo repetido. Por ejemplo, en lugar de

10 FORMAT (F5.3, F2.1, F5.3, F2.1,
*F5.3, F2.1)

se puede escribir

10 FORMAT (3 (F5.3, F2.1))

Ejercicio 28. Escribir los operadores para la salida de las variables A, IB y C, que contienen los números 73.25, — 37235, 0,01272.

Además de las especificaciones numéricas Im, Fm.n y Em.n existen las especificaciones de redacción mX, mH, que se utilizan para ordenar la disposición de los nú-

meros en el papel y acompañarlos con comentarios.

La especificación mX sirve para omitir m posiciones, es decir, en el papel habrá m blancos. Por ejemplo, supongamos que A = − 15., B = 0.45. Estas variables se pueden extraer con ayuda de los operadores

```
    PRINT 10, A, B
10 FORMAT (5X, F4.0, 5X, F4.2)
```

En el papel los números se colocarán así:

␣␣␣␣␣−15.␣␣␣␣␣0.45

La especificación mH permite trasladar al papel m símbolos, que siguen a la letra H. Se obtendrá el mismo resultado si estos m símbolos se toman en apóstrofos.

Ejemplo. Los operadores

```
    PRINT 10, A, B
10 FORMAT (5X, 2HA =, F4.0, 5X,
  *2HB=, F4.2)
```

o bien

```
    PRINT 10, A, B
10 FORMAT (5X, 'A=', F4.0, 5X,
  *'B=', F4.2)
```

imprimirán los números A y B en la forma siguiente:

␣␣␣␣␣A = −15. ␣␣␣␣␣B =
= 0.45

La lista de las especificaciones jugó siguiente papel: 5X — se omitieron 5 poel siciones; 2HA = — en el papel se han

trasladado dos símbolos después de la letra H, es decir, A=; F4.0 — se imprimió el valor de A; 5X — se omitieron 5 posiciones; 2HB = — se trasladaron al papel dos símbolos B=; F4.2 — se ha imprimido el valor de B.

Utilizando solamente las especificaciones de redacción se pueden hacer salir dibujos a la impresora (véanse ejemplos en el capítulo IV).

Ejemplo. Escribir un programa para el cálculo del producto de los números $a = 1{,}5873956$ y $b = 3{,}5$.

```
      PROGRAM N2A
      A = 1.5873956
      B = 3.5
      C = A*B
    PRINT 2,C
    2 FORMAT (10X, 4HA*B=, F10.8)
      END
```

En el papel, desde la 11ª posición, se imprimirá lo siguiente:

```
    A*B = 5.55588460
```

Ejercicio 29. 1. Escribir un programa de cálculo del área de un triángulo con base $a = 15{,}7328$ y altura $h = 0{,}031289$.

2. Escribir un programa de cálculo de la suma de los primeros diez términos de una progresión geométrica desde $b_1 = 1{,}135619$ y $q = 0{,}999999$.

En el Fortran existen varios símbolos que ordenan el trabajo del dispositivo impresor cuando estos símbolos se extraen a la impresora en cualquier renglón como los

primeros. Semejantes especificaciones administradoras en el Fortran son varias:

1H0, omisión de un renglón;

1H1, salto a una página nueva;

1H4, salto a la siguiente mitad de la página;

1H5, salto al siguiente tercio de la página;

1H6, salto al siguiente cuarto de la página;

1H+, impresión sin salto a un renglón nuevo, es decir, superponer los símbolos del renglón nuevo a los símbolos del renglón viejo;

1HS, bloqueo del salto automático a una página nueva durante la impresión del texto.

Durante la impresión tiene lugar el paso automático de una página a otra después de un número determinado de renglones. Con esto entre las páginas se obtiene un blanco. Si se hace salir a la impresora un gráfico o un dibujo, este blanco, frecuentemente, desfigura el cuadro. En estos casos, al comienzo de cada renglón, se extrae el símbolo S.

Señalaremos que los símbolos 0, 1, 4, 5, 6, +, S dirigen el trabajo de la unidad impresora solamente cuando son los primeros símbolos en el renglón. En este caso ellos mismos no salen impresos.

El programador debe recordar que los *símbolos precitados siempre dirigen la impre-*

sora si son los primeros que salen en el renglón. Por esto, si en la primera posición se extrae un número que comienza por 0, 1, 4, 5, 6 o por el signo + entonces el dispositivo de impresión reaccionará obligatoriamente al símbolo de salida (por ejemplo, saltará una parte de la página).

En estos casos el programador inexperto afirma que «la impresora se ha roto».

Para asegurarse contra semejantes contrariedades el nuevo renglón debe comenzar siempre por un blanco, es decir, al extraer los números en la lista del operador FORMAT debe figurar en primer término la especificación nX.

En el Fortran de las EC hay solamente 3 símbolos que dirigen el trabajo de la impresora: 0, 1 y +. Significan, respectivamente, omitir un renglón, saltar a una página nueva e imprimir sin saltar a un renglón nuevo.

Perforación de las tarjetas (fichas). El programa escrito puede ser horadado en tarjetas para su introducción en el ordenador electrónico. En la tarjeta existen 80 columnas (posiciones). Los operadores del Fortran se perforan en las columnas 7—72. Las columnas 1—5 se asignan para la marca del operador. En las columnas 73—80 se puede alojar cualquier comentario, por ejemplo, el número de la ficha perforada. Si el operador del Fortran no cabe en una ficha entonces su continuación se perfora en la tarjeta siguiente, en la que en la 6ª posición se pone el signo de continua-

ción: cualquier símbolo excepto cero. Puede haber varias tarjetas de continuación.

Así, el destino de las posiciones de la tarjeta al ser perforada con el programa Fortran es el siguiente:

Posición	Destino
1—5	Marcas de los operadores
6	Símbolo de continuación (por ejemplo, A o bien*)
7—72	Operador del Fortran (o su continuación)
73—80	Comentarios (por ejemplo, el número de la tarjeta)

Operador READ. Para la introducción de números que se utilizan en calidad de datos iniciales, sirve el operador READ, que se escribe en la forma

READ n, A, B, . . ., X

Aquí n es la marca del operador FORMAT, A, B, . . ., X, los nombres de las variables a las que se atribuyen los valores que se introducen.

En el operador FORMAT, que trabaja conjuntamente con READ, se utilizan las mismas especificaciones (Im, Em. n, Fm.n, nX, nH) que al trabajar con el operador PRINT.

Supongamos, por ejemplo, que los coeficientes de la ecuación cuadrática $Ax^2 +$

$+ Bx + C = 0$ para el programa del cálculo del discriminante están perforados en una tarjeta así: A, en las cuatro primeras columnas; B, en las cuatro siguientes (desde la 5ª hasta la 8ª); C, en las cinco siguientes (desde la 9ª hasta la 13ª). Los valores de A, B y C son, respectivamente, iguales a 1.32, 2.75 y — .345. Entonces a estos números les corresponderán las especificaciones de entrada F4.2, F4.2 y F5.3. Para la introducción de estos números sirven los operadores

```
READ 5, A, B, C
5 FORMAT (F4.2, F4.2, F5.3)
```

Si los números A, B y C se colocan de otra manera en la tarjeta entonces cambiarán también las especificaciones en el operador FORMAT. Por ejemplo, se pueden perforar los números de tal manera que estén separados entre sí por un blanco: 1.32␣2.75␣—.345␣. La posición vacía se puede relacionar con el número anterior y considerarla como una cifra más después del punto (igual a cero). De tal modo, el operador FORMAT se escribirá de la manera siguiente

```
5 FORMAT (F5.3, F5.3, F5.3)
```

Igual que durante la impresión, las especificaciones que se repiten pueden no escribirse reiteradamente, utilizando el coeficiente de repetición:

```
5 FORMAT (3F5.3)
```

Escribamos ahora una nueva variante del programa DISKR: sea así que los valores A, B y C se han perforado en la ficha por la especificación F5.3

```
      PROGRAM DISKRL
      READ 2, A, B, C
    2 FORMAT (3F5.3)
      D=B**2-4.*A*C
      PRINT 3, D
    3 FORMAT (3X, F5.2)
      END
```

En el ejemplo examinado todos los números introducidos han sido perforados en una ficha. Si cada uno de los números a introducir se perfora en una ficha aislada, entonces las especificaciones de estos números en el operador FORMAT estarán separadas unas de otras no por una coma, sino por una raya oblicua:

```
    5 FORMAT (F5.3/F5.3/F5.3)
```

o bien

```
    5 FORMAT (3 (F5.3/))
```

Examinemos el caso cuando en la lista del operador READ hay más variables que especificaciones numéricas (Im, Fm.n, Em.n) en el operador FORMAT. Por ejemplo,

```
      READ 2, A, B, C
    2 FORMAT (F5.2, F4.2)
```

En este caso se supone que dos números han sido perforados en una tarjeta por las especificaciones F5.2 y F4.2, y el tercer número ha sido perforado en la tarjeta siguiente por la especificación F5.2; la lista de las especificaciones del operador FORMAT se revisa de nuevo, comenzando desde el último abrir paréntesis. En el ejemplo

```
READ 3, A, B, C, D, E, F
3 FORMAT (F5.2, 2 (F3.2, F7.3))
```

cinco número deben ser perforados en una tarjeta por las especificaciones F5.2, F3.2, F7.3, F3.2, F7.3, y el sexto número en la tarjeta siguiente por la especificación F3.2. Para introducir los números *A*, *B* y *C*, perforados en fichas aisladas por una misma especificación F5.3, es suficiente escribir

```
READ 10, A, B, C
10 FORMAT (F5.3)
```

Al perforar números en las fichas se pueden ocupar todas las 80 posiciones. En el operador FORMAT se debe prover el paso a una nueva tarjeta cada vez que sean agotadas las 80 posiciones de la ficha en turno. Por ejemplo, el operador

```
25 FORMAT (5F15.6,I8)
```

exige ocupar en una ficha más de 80 posiciones ($5 \cdot 15 + 8 = 83$) y, por lo tanto, es erróneo.

Funciones matemáticas elementales. Para el cálculo de funciones como son sen x, ln x, $|x|$, $\sqrt{x}$ y otras existen progra-

mas estándares compuestos con anticipación. Para hacer uso de este programa es suficiente indicar el nombre de la función correspondiente y, entre paréntesis, escribir el valor del argumento en forma de una expresión aritmética arbitraria, por ejemplo SIN (3.14/6).

Daremos una tabla de los nombres de las funciones más empleadas (véase también el anexo 2):

Función	Escritura en el Fortran	Función	Escritura en el Fortran
sen x	SIN(X)	$\sqrt{x}$	SQRT(X)
cos x	COS(X)	$\lvert x \rvert$	ABS(X)
ln x	ALOG(X)	e^x	EXP(X)
log x	ALOG 10(X)		

Para las funciones trigonométricas el ángulo debe darse en radianes.

Ejemplo. Programemos en el Fortran la fórmula $y = ab\sqrt{a} + \ln \operatorname{sen} (t + x)$:

```
Y = A*B*SQRT (A) + ALOG (SIN (T+X))
```

Ejemplo. Componer un programa para el cálculo de la longitud L de la circunferencia si se conoce la superficie S del círculo correspondiente. Supongamos que $S < 400$ y que este número está perforado en la ficha con dos signos después del punto.

Escribamos la fórmula que expresa L mediante S:

$$L = 2\pi R = 2\pi \sqrt{\frac{S}{\pi}} = 2\sqrt{\pi S}.$$

Escribamos el programa.

```
    PROGRAM L
    READ 10, S
10  FORMAT (F6.2)
    A = 2.*SQRT (3.14*S)
    PRINT 20, A
20  FORMAT(10X, 12HLONGITUD DE
   *18HLA CIRCUNFERENCIA =,
   * F5.2)
    END
```

Para la introducción de S se tomó la especificación F6.2, ya que este número puede ocupar seis posiciones, dos de las cuales están después del punto. Una vez calculada la longitud de la circunferencia A la máquina imprimirá el resultado. Las primeras dos especificaciones de la extracción son de redacción: 10X, para retroceder a la derecha en diez posiciones, y 12H ... saca a la impresora el comentario «la longitud de la circunferencia =». Luego será imprimido el valor de A por la especificación F5.2, tomada por las siguientes razones: si $S < 400$ entonces $\sqrt{S} < 20$, $A = 2\sqrt{\pi} \cdot \sqrt{S} < 80$, es decir, la parte entera de A contiene no más de 2 cifras. Si el orden del resultado se calcula de manera no tan

simple, entonces se debe emplear la especificación Em.n.

En el ejemplo expuesto la longitud de la circunferencia se designa por A, en lugar de la admitida L, pues en el Fortran el nombre L indica una variable entera y por consiguiente, una vez calculada $L = 2\sqrt{\pi S}$ todos los signos decimales serían omitidos.

Ejercicio 30. 1. Componer el programa de cálculo de las raíces de la ecuación cuadrática si los coeficientes a, b y c han sido perforados en una misma ficha, yacen en el diapasón $[-10, 10]$ y se dan con dos signos decimales después del punto. Se supone que los números a, b y c se dan de tal manera que $b^2 - 4ac \geqslant 0$.

2. Componer un programa de cálculo del área y del perímetro de un triángulo conociendo dos lados a, b y el ángulo α entre ellos. Los valores a, b y α (en radianes) se han perforado en una ficha con 4 signos después del punto, y además a y b no superan 100.

3. Escribir en Fortran las fórmulas

$$\text{a)}\ y = \sqrt{ab}\,\operatorname{sen} a \ln b + |\operatorname{sen} a| \sqrt{|\operatorname{sen}^2 \alpha - \cos^2 b|},$$

$$\text{b)}\ y = \frac{a}{2\sqrt{b\sqrt{a}}}.$$

Operador CONTINUE. Este operador es el portador de una marca y no cumple ninguna otra función. Su aspecto general es

m CONTINUE

donde m es la marca.

Ejemplo. Supongamos que se debe componer un programa de tal manera que la salida de éste (terminación del trabajo)

se efectúe después de los operadores situados en distintos lugares del programa. Entonces, al final del programa, se puede escribir el operador 5 CONTINUE, y después de cada uno de los tres operadores mencionados poner GO TO 5.

```
     PROGRAM C
     ...
     GO TO 5
     ...
     GO TO 5
     ...
   5 CONTINUE
     END
```

Es particularmente cómodo utilizar el operador CONTINUE en el proceso de la escritura del programa, cuando se indica el paso a la parte del programa todavía no escrita por el programador. Supongamos, por ejemplo, que en el programa se indica el salto GO TO 17, pero todavía no hemos escrito los operadores que corresponden al curso sucesivo de la solución. Al escribir 17 CONTINUE nosotros, en primer lugar, podemos obtener el comienzo del programa en forma definitiva (es decir, indicar en GO TO una marca concreta) y, en segundo lugar, estaremos garantizados de que la marca 17 no será omitida en el programa.

El operador CONTINUE es cómodo porque permite, en caso de necesidad, reunir las funciones de varios operadores. Supongamos, por ejemplo, que en el programa se han escrito los operadores GO TO 17 y

GO TO 36, y que en lo sucesivo se aclaró que los operadores con marcas 17 y 36 deben tener un mismo destino. Entonces se pueden poner seguidamente dos operadores CONTINUE

```
   ...
17 CONTINUE
36 CONTINUE
   ...
```

Otros casos de utilización de este operador se expondrán en los programas sucesivos.

Operador de salto convencional IF. Al resolver la ecuación cuadrática $ax^2 + bx + c = 0$ pueden presentarse dos casos: el discriminante $D = b^2 - 4ac \geqslant 0$ o bien $b^2 - 4ac < 0$. Antes de extraer la raíz del discriminante se debe investigar su signo y, en dependencia de éste, proceder de uno de los dos métodos: o bien calcular las raíces $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ o bien declarar que la ecuación dada no tiene raíces reales.

En el lenguaje Fortran existe un operador de salto convencional IF, que ejerce la elección de la operación siguiente en dependencia de cierta condición. El aspecto general de este operador

```
IF (E) n, m, k
```

Aquí n, m, k son las marcas de algunos operadores del programa, E, una expresión aritmética (en particular, nombre de la variable).

El operador IF (E) n, m, k funciona así: se calcula el valor de la expresión E y se efectúa el salto al operador con una de las marcas n, m y k de acuerdo a la regla siguiente:

1) si $E < 0$, entonces a la marca n,
2) si $E = 0$, entonces a la marca m,
3) si $E > 0$, entonces a la marca k.

Por ejemplo, el operador

```
IF (X**2—5.) 15, 18, 31
```

efectúa el salto al operador con la marca 15, si X**2—5. < 0 al operador con la marca 18, si X**2—5 = 0, y al operador con la marca 31 si X**2—5. > 0.

En dependencia del sentido de la condición dos marcas indicadas en el operador IF pueden coincidir. Por ejemplo, al resolver una ecuación cuadrática los casos $D > 0$ y $D = 0$ se pueden unificar, siendo así que el operador IF se escribirá como

```
IF (B**2—4.*A*C) m, k, k
```

Si nos interesa solamente el caso de raíces iguales, es decir, $D = 0$, entonces el operador IF correspondiente se escribirá así:

```
IF(B**2—4.*A*C) m, n, m
```

Ejemplo. Programa de solución de una

ecuación cuadrática para un valor arbitrario del discriminante.

```
      PROGRAM SQEQU
      READ 1, A, B, C
    1 FORMAT (3F9.6)
      D = B**2—4.*A*C
      IF(D)2, 3, 3
    3 X1 = (—B+SQRT(D))/(2.*A)
      X2 = (—B—SQRT(D))/(2.*A)
      PRINT 4, X1, X2
    4 FORMAT (5X, 3HX1 =, F12.6/5X
     3HX2 =, F12.6)
      GO TO 6
    2 PRINT 5
    5 FORMAT (5X, 10HRAICES␣NO␣
     *3HHAY)
    6 CONTINUE
      END
```

En caso de $D \geqslant 0$ se imprimirán los valores de las raíces en la forma siguiente:

␣␣␣␣␣X1 = KKKKK. nnnnnn
␣␣␣␣␣X2 = KKKKK. nnnnnn

Aquí K indica las posiciones destinadas para el alojamiento de la parte entera, inclyuendo el signo, n son las posiciones para la parte fraccionaria. En el caso de

D $<$ 0 se imprimirá el texto

RAICES␣NO␣HAY

Ejercicio 31. Escriban en Fortran el algoritmo de cálculo de la función $y = |a|$, sin utilizar la función ABS(X).

Ciclos. En algunos problemas resulta ser necesario realizar varias veces una misma secuencia de operaciones.

Ejemplo. Calcular las áreas de cinco triángulos por tres lados.

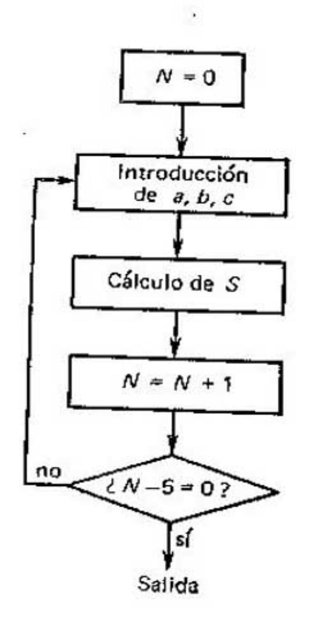
En el programa, para resolver semejante problema, es necesario repetir cinco veces la ejecución de un grupo de operadores:

```
   PROGRAM S5                          ⎤El grupo de
   READ 2, A, B, C                     ⎥operadores
   P = (A+B+C)/2.                      ⎥se repite
   S = SQRT(P*(P—A)*(P—B)*             ⎦5 veces
  *(P—C))
   PRINT 3, S
 3 FORMAT (5X, 2HS=, F12.3)
 2 FORMAT (3F6.3)
   . . .
   READ 2, A, B, C
   P = (A+B+C)/2.
   S = SQRT(P*(P—A)*(P—B)*(P—C))
   PRINT 3, S
   END
```

Para simplificar la escritura de este programa se utiliza el algoritmo siguiente. Originariamente se atribuye el valor 0 a cualquier variable N denominada contador.

Después de cada ejecución de un grupo de operadores repetidos, a N se añade 1 y se comprueba la condición N = 5. Cuando se cumpla esta condición el proceso concluye.

El esquema de solución del problema (esquema en bloque) es:



La escritura de este algoritmo en el Fortran es:

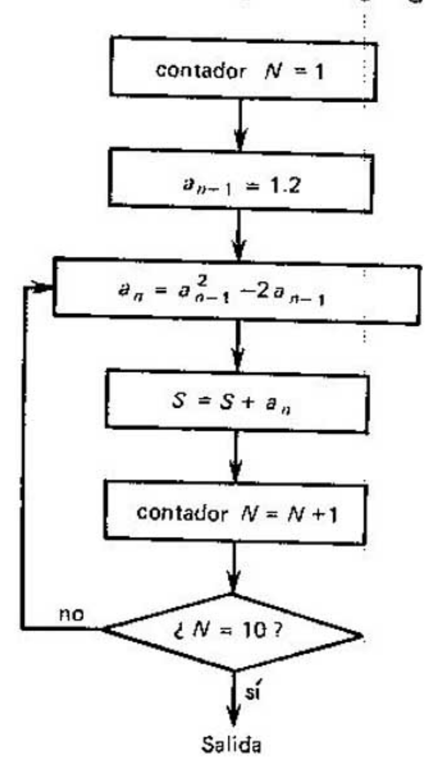
```
PROGRAM N6
N = 0                      Borrado del contador
```

30 READ 2, A, B, C	Introducción de números
2 FORMAT (3F6.3)	
P = (A+B+C)/2.	Cálculo del semiperímetro
S=SQRT(P*(P—A)* *(P—B)*(P—C))	Fórmula de Herón
PRINT 3, S	Impresión de los valores de S
3 FORMAT(5X, 2HS=, *F12.3)	
N = N+1	Adición de 1 al contador
IF (N—5)30, 4, 4	Comprobación: ¿N=5?
4 CONTINUE	Si N=5, entonces,
END	fin del problema

Las partes del programa que se realizan múltiplemente se denominan *ciclos*. En el ejemplo la parte del programa desde el operador con la marca 30 hasta IF(N—5)30, 4, 4 es un ciclo.

Ejemplo. Calcular la suma de los primeros diez términos de la sucesión $a_n = a_{n-1}^2 + 2a_{n-1}$; $a_1 = 1{,}2$.

El esquema en bloque del programa es:



Al contador se envía 1, y no 0, pues en el ciclo el cálculo comienza por a_2.

En el Fortran el programa es:

PROGRAM N7	
N = 1	Registro de 1 en el contador
S = 1.2	Valor inicial de S
AN = 1.2	$a_1 = 1{,}2$

```
 1 AN = AN**2-          Fórmula recurrente
 *AN*2                   para el cálculo de a_n
   S = S+AN              Adición a S de la
                         consecutiva a_n
   N = N+1               Adición de 1 al con-
                         tador
   IF(N-10)1, 2, 2       Prueba: ¿N=10?
 2 CONTINUE              Si N=10
   PRINT 3, S            impresión del valor S
 3 FORMAT (5X, 2HS=
 *F12.3)
   END
```

La parte desde el operador con la marca 1 hasta IF(N—10)1,2,2 es un ciclo.

Ejemplo. Calcular K = 11!

El programa en el Fortran es:

```
   PROGRAM FACT
   K=1                   Valor inicial del fac-
                         torial
   N=0                   Borrado en el con-
                         tador
 1 N = N+1               Aumento del conta-
                         dor en 1
   K = K*N               Cálculo de K!
   IF(N-11)1, 2, 2       Prueba: ¿N=11?
 2 CONTINUE              Si N=11
   PRINT 3, K            imprimir los valores
                         de K!
 3 FORMAT (7X,
 *2HK=, I10)
   END
```

Operador DO de ciclo. En el Fortran existe un operador especial DO de ciclo. El

aspecto general del operador es

DO n I = m, j, k

donde n es la marca del último operador del ciclo; I es una simple variable entera que es el parámetro (contador) del ciclo; m es el valor inicial del parámetro del ciclo I; j es el límite superior de I: la salida del ciclo tiene lugar cuando I > j; k es el paso por los valores del parámetro del ciclo al repetir la ejecución.

Aquí m, j y k pueden ser constantes enteras sin signo o simples variables enteras que adquieren valores positivos.

Se repite el grupo de operadores, el primero de los cuales sigue después de DO, y el último tiene la marca n.

Ejemplo.

```
DO 5 I = 2, 11, 3
A = 1.
...
5 CONTINUE
```

Se repetirán los operadores desde A = 1. hasta 5 CONTINUE cuatro veces: para I = 2, I = 5, I = 8, I = 11. Si el paso por I es igual a 1 entonces el operador DO se puede escribir en forma

DO n I = m, j

(*k* se omite conjuntamente con la coma anterior).

Utilizando el operador DO del ciclo se puede simplificar la escritura del programa FACT:

```
      PROGRAM FACT1
      K=1
      DO 1 L=2, 11 ]    el ciclo se repite
      K = K*L      ]    10 veces
    1 CONTINUE
      PRINT 3, K
    3 FORMAT (7X,
     *2HK=, I10)
      END
```

Ejemplo. Hallar el 20º término de la sucesión

$$a_n = a_{n-1}^2 + 0{,}01; \ a_1 = 1.$$

Dos variantes del programa en el Fortran:

```
      PROGRAM N9               PROGRAM N9A
      A=1.
      DO 1 K = 2, 20           A=1.
      A = A**2+0.01            DO 1 K=2, 20
    1 CONTINUE               1 A=A**2+0.01
      PRINT 2, A               PRINT 2, A
    2 FORMAT (10X,           2 FORMAT (10X,
     *2HA=, F6.2)             *2HA=, F6.2)
      END                      END
```

El parámetro del ciclo es una variable entera, es decir, su nombre debe comenzar por una de las letras I, J, K, L, M, N y contener no más de seis letras y cifras. Si en el operador DO n I = m, j, k la magnitud j se da por una constante, entonces ésta no puede exceder de un valor determinado para cada tipo de ordenador electrónico.

Se puede utilizar un ciclo en otro ciclo. En este caso se habla del ciclo exterior y del interior, situado enteramente dentro del ciclo exterior. En el ciclo interno no se puede utilizar el parámetro del ciclo externo. Por ejemplo:

```
   DO 3 N3 = 5, 10
   ...
   DO 2 N2 = 1, 20
   ...
   DO 1 N1 = 3, 100
   ...
 1 CONTINUE
   ...
 2 CONTINUE
   ...
 3 CONTINUE
```

El ciclo exterior por el parámetro N3 termina con el operador 3 CONTINUE. Este ciclo comprende en sí el ciclo por N2, que termina con el operador 2 CONTINUE. El ciclo más interno por el parámetro N1 termina con el operador 1 CONTINUE.

Este fragmento del programa funciona así:

1) se ejecutan todos los operadores hasta 1 CONTINUE, y luego el ciclo interno concluye totalmente su trabajo (N1 varía desde 3 hasta 100);

2) se ejecutan los operadores hasta 2 CONTINUE, es decir, funciona 20 veces el ciclo por N2, que incluye en sí el funcionamiento en un múltiplo de 98 del funcionamiento del ciclo por N1;

3) se ejecutan los operadores hasta 3 CONTINUE, es decir, el ciclo exterior funciona 6 veces. Con esto, el ciclo por N2 se repetirá 6·20 veces, y el ciclo por N1 se repetirá 6·20·98 veces.

Ejemplo. Hallar el número de billetes «afortunados» con los números desde 000000 hasta 999999 inclusive. El billete se considera «afortunado» si la suma de las tres cifras izquierdas del número es igual a la suma de las tres cifras de la derecha. Vamos a resolver el problema revisando todos los números posibles, es decir, variando cada cifra del número desde 0 hasta 9.

Supongamos que el número está registrado en la forma IJKLMN. El programa de solución de este problema aparece así:

PROGRAM HAPPY	
NC=0	NC es el número de billetes «afortunados»; la revisión de todas las combinaciones en un número de seis cifras
DO 1 I=1, 10	I, centenares de miles
DO 1 J=1, 10	J, decenas de miles
DO 1 K=1, 10	K, miles
DO 1 L=1, 10	L, centenas
DO 1 M=1, 10	M, decenas
DO 1 N=1, 10	N, unidades
IF(I+J+K— •(L+M+N))1, 2, 1	Prueba: ¿es «afortunado» el billete?
2 NC=NC+1	Si es afortunado entonces el número NC se aumenta en 1

```
1 CONTINUE
  PRINT 3, NC          Impresión del número de los billetes «afortunados»
3 FORMAT (10 X,
 *9HBILLETES□
 *11HAFORTUNADOS, I6)
 END
```

Advertiremos que los valores de los parámetros (I, J, K, L, M, N) varían desde 1 hasta 10, mientras que las cifras de los correspondientes órdenes de los números deben variar desde 0 hasta 9. Pero semejante desplazamiento no influirá sobre el resultado: las tres unidades excedentes de la suma I + J + K se aniquilarán por las tres unidades del substraendo (L + M + + N). En el ejemplo expuesto se han utilizado seis ciclos locales. El Fortran admite un número suficientemente grande de encajes de este tipo.

Al escribir ciclos es necesario atenerse a las reglas siguientes:

1. En el ciclo se puede entrar solamente mediante el operador DO.

2. Se puede concluir el ciclo con cualquier operador, excepto IF, GO TO, RETURN, DO, STOP.

3. Dentro del ciclo dado DO no puede acabarse el ciclo DO, exterior respecto a él.

4. Se pueden utilizar hasta 50 ciclos encajados uno en otro.

5. Los últimos operadores de los ciclos encajados pueden coincidir.

Ejemplo.

```
DO 5 I = 1, 7, 2
DO 5 K = 5, 10, 3
DO 5 L = 3, 5
...
5 CONTINUE
```

6. Dentro del ciclo no se puede cambiar el valor del parámetro de éste (inicial y final) y el paso.

7. Después de concluido el ciclo no se puede utilizar el valor de su parámetro.

La salida del ciclo se puede realizar por cualquier condición.

Ejemplo. Calcular la suma de la serie $S = 1 - 1/2! + 1/3! - 1/4! + \ldots$, con exactitud de hasta 0,01. Esto significa que el proceso debe ser interrumpido en el sumando cuya magnitud absoluta $<0,01$.

PROGRAM N11	
S=1.	Valores iniciales de S y A
A=1.	
EPS=0.01	Se da la exactitud de cálculo
DO 2 I=2, 10	Valor final del parámetro (10), a ciencia cierta es mayor del que se requiere
A=−A/I	Durante el primer paso del ciclo A=−1/2, durante el segundo, A=+1/(2·3), etc.

`IF(ABS(A)—EPS)` `*1, 2, 2`	Si $\|A\| \geqslant$ EPS entonces el proceso no ha terminado;
`2 S=S+A`	se calcula la suma de la serie; este operador es el último del ciclo: el siguiente (si $I \leqslant 10$) a cumplir es el operador A=—A/I
`1 CONTINUE`	La salida a este operador sucede con la condición $\|A\| <$ EPS ó $I \geqslant 10$
`PRINT 3, S`	Salida de la suma de la serie a la impresora

`3 FORMAT (10X,`
`*2HS=, F6.3)`
`END`	

Ejercicio 32.

1. Hallar el 15-avo término de la sucesión $a_k = \sqrt{a_{k-1}} + 1$ si $a_1 = 1$.

2. Hallar la suma de veinte términos de la sucesión $a_n = \frac{a_{n-1}^2}{a_{n-1} + 3}$, si $a_1 = 2$.

1.9.1. Colecciones de variables

Frecuentemente es cómodo reunir varias variables en una colección. A la colección o fichero se le da un nombre, por ejemplo B. Toda variable que entra en esta colección (elemento de la colección) lleva el mismo nombre B, y su número ordinal en la colección se señala entre paréntesis y se

denomina índice. Semejantes variables se llaman variables con índice, a diferencia de las variables simples que no tienen índice, es decir, que no son elementos de la colección.

Supongamos que B es una colección que reúne 5 variables cuyos valores numéricos son 1.6, 1.1, 2.2, 1.8, 3.05. Entonces la variable B(3) tiene el valor numérico 2.2; B(5) tiene el valor 3.05. Semejantes colecciones, cuyos elementos se determinan por un índice se denominan unidimensionales. En este capítulo examinaremos solamente las colecciones unidimensionales.

Operador DIMENSION. Para que el compilador conceda un número determinado de células a la colección es necesario después del operador PROGRAM escribir el operador

DIMENSION B(M)

donde B es el nombre de la colección y M, el número de elementos en ella. Aquí M es una constante entera sin signo.

En el ejemplo expuesto la colección B contiene 5 elementos, por lo que para la descripción de B se utiliza el operador

DIMENSION B(5).

Para la extracción a la impresora o para la introducción de toda la colección es suficiente indicar su nombre en los operadores PRINT o READ. Por ejemplo, para la extracción de la colección A compuesta de

1000 elementos, es suficiente escribir

PRINT 16, A

Si se necesita extraer solamente el N-ésimo elemento de la colección, entonces en el operador PRINT es menester indicar A(N). Por ejemplo, para extraer el 5° elemento de la colección A el operador de impresora tiene el aspecto siguiente:

PRINT 16, A(5)

Se debe prestar atención a que la escritura A(10) en el operador DIMENSION A(10) significa una colección compuesta de 10 elementos, mientras que en los operadores restantes que estudiamos esta escritura significa el décimo elemento de la colección.

El operador DIMENSION existe en el programa solamente para comunicar al compilador cuántas células se deben ceder a la colección, es decir, cómo se debe distribuir la memoria de la máquina. En el programa traducido no hay instrucciones de máquina que correspondan a este operador. Semejantes operadores del Fortran se denominan *operadores que no se ejecutan* o *declaratorios.*

En un operador DIMENSION se puede dar información respecto a varias colecciones.

DIMENSION A(50), B(100), C(30)

CAPÍTULO II

Para aquellos que decidieron avanzar más en sus conocimientos

Habiendo estudiado el primer capítulo el lector ha obtenido los conocimientos acerca de los programas simples. En este capítulo continuaremos la referencia respecto a la programación para aquellos que quieren resolver problemas bastante complicados.

Además, daremos algunas nociones sobre el funcionamiento del compilador y daremos a conocer al lector ejemplos aleccionadores de programas.

2.1. Conocimientos complementarios sobre el lenguaje Fortran

2.1.1. Cálculos de elevada exactitud

Como sabemos, para cada número se asigna, generalmente, una célula de la memoria. La exactitud máxima de presentación de los números por la máquina queda determinada por las particularidades constructivas del ordenador electrónico del tipo dado.

En una serie de casos se requiere efectuar cálculos de mayor precisión. En semejantes

casos se recurre a los cálculos de *doble precisión* es decir, para cada cifra se asignan dos células consecutivas de la memoria, en lugar de una. En el lenguaje Fortran semejante representación de los números corresponde al tipo DOUBLE PRECISION, es decir, con *precisión doble*.

Las constantes con doble precisión se inscriben en la forma K.nD±L o K.D±L. Aquí K es la parte entera de la constante, n, la parte fraccionaria, D, el índice de precisión doble, y ±L es el exponente de la potencia con base 10. La parte entera puede no existir, y el signo + en el exponente de la potencia puede ser omitido.

Ejemplo. El número e = = 2,718 281 828 459 045 se puede escribir en la forma 2.718 281 828 459 045 D0 o bien 0.271 828 182 845 9045 D1 o bien 002 718 281 828 459 045 D3.

Las magnitudes variables que tienen tipo de «doble precisión» se describen con el operador DOUBLE PRECISION.

Ejemplos.

DOUBLE PRECISION A, XT7, PI, NK(8)

DOUBLE PRECISION BPRIM(5), ENTRY

En cada operador DOUBLE PRECISION se pueden indicar algunas variables o colecciones, cuyos nombres se deben separar con comas.

Si se necesita describir la colección con doble precisión se puede indicar en el ope-

rador DOUBLE PRECISION el nombre de esta colección conjuntamente con la dimensión. En este caso no se necesita el operador DIMENSION.

Se puede, no obstante, indicar en el operador DOUBLE PRECISION solamente el nombre de la colección (sin dimensión). En este caso, para la descripción completa de la colección es necesario también el operador DIMENSION, en el que se indica el nombre de la colección y su dimensión.

Por ejemplo, la colección X(15) del tipo «de doble precisión» puede ser descrita por cualquiera de los tres procedimientos:

a) DOUBLE PRECISION X(15)
b) DOUBLE PRECISION X
   DIMENSION X(15)
c) DIMENSION X(15)
   DOUBLE PRECISION X

El operador DOUBLE PRECISION es un operador declarativo. Éste debe preceder a cualquier operador a ejecutar y puede disponerse en orden arbitrario respecto a otros operadores declarativos (en particular, al DOUBLE PRECISION y a los operadores DIMENSION).

Si en el programa se utiliza una constante, una variable o una colección del tipo DOUBLE PRECISION ello significa, en primer lugar, que para el alojamiento de cada una de las magnitudes de este tipo se designarán dos células de la memoria y, en segundo lugar, que las operaciones aritmé-

ticas con ellas se realizarán por reglas diferentes que, digamos, para las magnitudes de tipo real.

2.1.2. *Cálculos con números complejos*

Durante la solución de una serie de problemas aplicados surge la necesidad de realizar cálculos con números complejos.

Todo número complejo $x + iy$ ocupa en la memoria de la máquina dos celdas consecutivas, en la primera de las cuales se encuentra la parte real x, y en la segunda, la parte imaginaria y de este número.

En el Fortran los números complejos se escriben de otra manera que en la matemática. En el Fortran al número complejo $x + iy$ le corresponde la escritura en forma de una constante compleja (x, y). Aquí x e y deben ser constantes *reales*. Por ejemplo, el número complejo $2{,}3—5i$ puede ser representado en la forma (2.3, —5.) ó (0.23E1, —5.00), pero no (2.3E1, —5).

La variable o la colección de tipo complejo se describen por el operador COMPLEX.

Ejemplos.

```
COMPLEX ART, VEGA(6), INTEGR
COMPLEX DRUM, VECTOR
DIMENSION VECTOR(18)
```

Aquí la colección compleja VEGA está descrita en el operador COMPLEX conjuntamente con la dimensión, y la colección compleja VECTOR está descrita en el operador COMPLEX sin la indicación de la

dimensión, mientras que en el operador DIMENSION se indican su nombre y dimensión.

El operador COMPLEX, igual que el operador DOUBLE PRECISION, es un operador declarativo y sirve solamente para comunicar al compilador que la variable dada o la colección tienen tipo complejo. De acuerdo con esta indicación respecto al tipo de la variable, el compilador asigna para el alojamiento de cada número semejante dos celdas de la memoria y forma las instrucciones de máquina, que realizan las operaciones aritméticas de acuerdo con las reglas establecidas para los números complejos.

Ejemplo.

```
COMPLEX RT, MAN
RT = (—2.5, 7.16)
MAN = RT*(12.3, —6.) * (1.7, 2.6)**2
```

2.1.3. Utilización de magnitudes de distintos tipos en una expresión aritmética

Antes ya vimos que en una misma expresión aritmética se pueden utilizar magnitudes de tipos real y entero. Al introducir dos tipos nuevos de magnitudes (DOUBLE PRECISION y COMPLEX) debemos determinar también las reglas de su utilización en una misma expresión aritmética.

Establezcamos las designaciones siguientes:

R, la magnitud del tipo real (REAL),

I, la magnitud del tipo entero (INTEGER),

C, idem del tipo complejo (COMPLEX),

DP, idem del tipo de precisión doble (DOUBLE PRECISION),

— — es la combinación inadmisible de los tipos.

Las tablas que a continuación se exponen demuestran (en la intersección del correspondiente renglón con la columna) el tipo del resultado que se obtiene al realizar operaciones aritméticas con las magnitudes de cualquiera de los cuatro tipos R,

El tipo de resultado para A+B, A—B, A*B, A/B

A \ B	R	I	C	DP
R	R	R	C	DP
I	R	I	C	DP
C	C	C	C	C
DP	DP	DP	C	DP

El tipo del resultado para AB**

A \ B	R	I	C	DP
R	R	R	—	DP
I	R	I	—	DP
C	—	C	—	—
DP	DP	DP	—	DP

I, C y DP. En las tablas las A se dan por la vertical, y las B por la horizontal.

En las tablas expuestas se ve que si la operación aritmética relaciona dos magnitudes de un mismo tipo el resultado será del mismo tipo (a excepción de las combinaciones inadmisibles de tipos). Si, por el contrario, la operación se realiza con dos magnitudes de distintos tipos entonces el tipo del resultado corresponde al tipo «superior» de las magnitudes que participan en la operación. Con esto, el rango de la superioridad de los tipos (desde el superior al inferior) es el siguiente: C, DP, R, I.

De la última tabla se deducen dos reglas:

1. *No se puede elevar a potencia compleja.*

2. *Las magnitudes complejas solamente se pueden elevar a una potencia entera.*

Ejemplo. Examinemos el ejemplo de la expresión aritmética que contiene operaciones con magnitudes de distintos tipos:

X + 2.5D0 — 5*(K + 2)**0.28

En el proceso del cálculo de esta expresión se obtendrán los siguientes resultados intermedios:

1) D1 = X + 2.5D0 (tipo DP),
2) I1 = K + 2 (tipo I),
3) R1 = I1**0.28 (tipo R),
4) R2 = 5*R1 (tipo R),
5) D2 = D1 — R2 (tipo DP).

El resultado tendrá el tipo DP, es decir, el tipo superior de los participantes en esta expresión.

2.1.4. Transformación del tipo de las magnitudes como resultado de la ejecución del operador de atribución

El operador de atribución contiene a la izquierda del signo = el nombre de la variable (simple o con índice), y a la derecha de este signo la expresión aritmética. El tipo de la variable puede no coincidir con el tipo de la expresión aritmética. Por ejemplo, en el operador K = X + 5. la variable K tiene el tipo entero, y la expresión X + 5. tiene tipo real.

Al ejecutar el operador de atribución el resultado se transforma siempre en el tipo de la variable que está a la izquierda del signo =. En el ejemplo estudiado el resultado, obtenido después del cálculo de la expresión X + 5., se transformará primero en un número de tipo entero y después se atribuirá a la variable K.

Ejemplos.

A = X + (1.5, —6.)*5 (resultado del tipo R)

N = 7./3. (resultado del tipo I, igual a 2)

T = 1.3 D1 + (3., 7.) (resultado del tipo R)

2.1.5. Operadores de descripción del tipo

Hasta ahora, al determinar el tipo de las magnitudes, utilizamos el siguiente acuerdo: si el nombre de la variable o de la

colección comienza por una de las letras I, J, K, L, M, N, entonces la variable (la colección) tiene tipo «entero», en caso contrario tiene tipo «real». Este acuerdo no se amplia, naturalmente, a las variables y colecciones descritas por los operadores DOUBLE PRECISION y COMPLEX.

En el Fortran se admite también otro procedimiento de descripción del tipo real y entero. Se puede describir el nombre de la variable correspondiente (o colección) en los operadores REAL o INTEGER. Por ejemplo, los operadores

```
INTEGER T, PSI(25), IVAN
REAL ANT, M50(50), J, TER
```

establecen el tipo entero para las variables T e IVAN y para la colección PSI (INTEGER), mientras que para las variables ANT, J y TER, y para la colección M50, el tipo real (REAL).

Si el nombre de la variable o de la colección no se indica en ninguno de los operadores REAL, INTEGER, COMPLEX o DOUBLE PRECISION, entonces a esta variable (colección) se le atribuye el tipo real o entero, de acuerdo con la regla respecto a la primera letra del nombre.

Los operadores REAL, INTEGER, COMPLEX y DOUBLE PRECISION se denominan *operadores de descripción del tipo*. Estos operadores son declarativos y se disponen al principio del programa antes del primer operador que se ejecuta. El orden

de disposición de estos operadores respecto a otros operadores declarativos es arbitrario.

Además de los cuatro tipos examinados, el lenguaje Fortran admite también otros tipos de magnitudes que no estudiaremos.

2.1.6. El operador DATA

Al escribir programas puede surgir la necesidad de inscribir en determinadas celdas de la memoria, antes del comienzo del funcionamiento del programa, los números requeridos.

Supongamos, por ejemplo, que calculamos el valor del polinomio $P = a_0 + a_1x + a_2x^2 + a_3x^3$, donde los coeficientes a_0, a_1, a_2 y a_3 son ciertos números dados. Claro, se puede atribuir a las variables correspondientes los valores dados, escribiendo, por ejemplo, 4 operadores de atribución. Sin embargo, cada uno de estos operadores, en primer lugar, ocupará cierta cantidad de células de la memoria, y, en segundo lugar, exigirá cierta cantidad de tiempo de máquina para su ejecución. En este caso es más cómodo inscribir los valores necesarios de las constantes en las células correspondientes de la memoria antes del comienzo del funcionamiento del programa. Semejante inscripción se efectúa con el operador DATA. Supongamos, por ejemplo, que queremos atribuir a los coeficientes del polinomio P los valores 1.38, —2.5E4, 18.3 y 3.6E2. Podemos designar estos coeficientes con los

nombres A0, A1, A2 y A3 y escribir el operador siguiente:

```
DATA A0, A1, A2, A3/1.38, —2.5E4,
*18.3, 3.6E2/
```

Aquí, al principio, se enumeran mediante comas los nombres de las variables en las que se deben introducir los números necesarios y luego, con rayitas inclinadas (/.../) se enumeran las constantes que deben ser inscritas.

Se pueden asociar los coeficientes A0, A1, A2 y A3 en una colección y escribir luego el operador DATA que inscribe en esta colección los valores de las cuatro constantes:

```
DIMENSION A(4)
DATA A/1.38, —2.5E4, 18.3, 3.6E2/
```

En el operador DATA se puede poner una coma, después de la rayita oblicua que lo cierra / e indicar después la nueva lista de variables y la correspondiente lista de constantes.

Por ejemplo,

```
DATA A0/1.38/, A1/—2.5E4/, A3/18.3/,
*A4/3.6E2/
```

Si entre las constantes que se inscriben existen repeticiones entonces, delante de la constante, se puede indicar el coeficiente de repetición y el signo *. Por ejemplo, el operador

```
DATA X1, Y1, Z1/1., 1., 1./
```

permite una inscripción más compacta

DATA X1, Y1, Z1 /3*1/.

El operador DATA, a diferencia del operador de atribución, es un operador que *no se ejecuta*. La inscripción de las constantes en las células de la memoria indicadas en el operador DATA, se efectúa por el sistema del software antes del comienzo del funcionamiento del programa.

Puesto que el operador DATA no se ejecuta debe situarse entre otros operadores que no se ejecutan del programa y preceder obligatoriamente al primer operador que se ejecuta.

Ejercicio 33. 1. Inscribir en la colección C compuesta de 8 elementos los números siguientes: 15.2, 17.3, 17.3, 17.3, 17.3, 17.3, 1.5, 3.2.

2. ¿Qué tipo de resultado se obtendrá en los casos siguientes:

DOUBLE PRECISION A, C
COMPLEX B, D
a) F = A*C+15
b) F = B**2—B+1.5
c) F = A**2—C+D?

2.1.7. El operador calculado GO TO

Nosotros ya hemos hecho conocimiento con el operador de salto GO TO m. La generalización de este operador es el GO TO calculado que se escribe en la forma

GO TO (m_1, m_2, . . ., m_k), L

Aquí m_1, m_2, . . ., m_k son las marcas de los operadores a los que se puede realizar

el paso, L, una variable simple entera, cuyo valor determina a dónde precisamente se efectúa el salto. Si en el momento de ejecución del GO TO calculado el valor de L es igual a 1 entonces sucederá el salto al operador con la marca m_1; si L = 2, al operador con la marca m_2, etc. El valor L determina, de este modo, el número ordinal de la marca a la que se efectúa el salto con ayuda del operador calculado GO TO.

Ejemplo.

```
LABEL = 3
. . .
GO TO (42, 7, 5, 28), LABEL
```

Aquí el control se transmitirá al operador con la marca 5, ya que a la variable LABEL se le atribuyó el valor 3, por lo que éste indica la tercera marca de la lista.

Para la ejecución del GO TO calculado es necesario que el valor de la variable L satisfaga la condición $1 \leqslant L \leqslant K$, donde K es el número de marcas en la lista. Si esta condición no se cumple entonces la marca no corresponderá al valor de L. En este caso la ejecución del operador transcurre diversamente, en dependencia del compilador.

Examinemos un ejemplo de utilización del operador GO TO calculado en el programa. Supongamos que tenemos un grupo de operadores que se utiliza en varios sitios del programa. Para no escribir este grupo tantas veces cuantas se encuentran en el programa se puede proceder de la manera

siguiente. El primer operador del grupo se anota con una marca. El operador GO TO calculado, en el que debe haber tantas marcas cuantas salidas posibles existen de este grupo de operadores, es el último del grupo que se inscribe.

```
   L=1
   GO TO 7
15 CONTINUE
   ...
   L=2
   GO TO 7
83 CONTINUE
   ...
   L=3
   GO TO 7
42 CONTINUE
   ...
 7 CONTINUE            ┐ grupo de operadores
   ...                 │ al que se recurre
   GO TO (15, 83,      │
  *42), L              ┘
```

Antes de utilizar el grupo de operadores

```
7 CONTINUE
  . . .
  GO TO (15, 83, 42), L
```

a la variable L se le atribuye el valor 1, ó 2, ó 3, según sea el número ordinal de la marca (15, 83, 42) a la que se transmitirá el control al terminar el trabajo del grupo de operadores.

2.1.8. El operador lógico IF

Nosotros ya conocemos el operador tipo IF (A) m, n, k, que, en dependencia de los valores de la expresión aritmética A, efectúa el paso a una de las tres marcas m, n, k. Este operador se denomina también IF *aritmético*.

Otra variedad del operador IF es el IF *lógico*. El aspecto general de este operador es

IF(L) S

donde L es una expresión lógica, y S, el operador.

La expresión lógica L puede adquirir uno de los dos valores: «verdad» o «mentira». En el caso cuando el valor de L es «verdad», después del operador IF lógico se ejecuta el operador S. En caso contrario (es decir, cuando el valor de L es «mentira»), el operador S no se ejecuta, y se efectúa la transmisión del control al operador que sigue al operador IF lógico. En calidad de operador S se puede coger cualquier operador, excepto DO e IF lógico.

Examinemos las expresiones lógicas más simples, las expresiones de la relación que más frecuentemente se utilizan en la práctica. El aspecto general de estas expresiones es: A1.mn.A2. Aquí A1 y A2 son las expresiones aritméticas, .mn., el signo de operación de la relación. Se permiten 6 operaciones de las relaciones: .EQ.— es igual, .NE.— no es igual, .GT.— es mayor,

.GE.— es mayor o igual, .LT.— es menor, .LE.— es menor o igual.

Ejemplo. $A > 5.3$ se inscribe como A.GT.5.3; $B < C$ como B.LT.C, $k^2/5 \leqslant \leqslant 25i$ como K**2/5..LE.25*I.

Ejemplo.

```
IF (K.NE.0) C = M/K*D
X = P + Q
. . .
```

Si la condición $K \neq 0$ es justa entonces, después del operador IF, se cumplirá el operador $C = M/K*D$, y después el operador $X = P + Q$. Si, por el contrario, $K = 0$ entonces después del operador IF se cumplirá inmediatamente el operador $X = P + Q$.

Ejemplo. Supongamos que se extrae la raíz de la magnitud C, que puede resultar ser negativa a cuenta de los errores de los cálculos. Es razonable asegurarse de semejantes contrariedades de la manera siguiente:

```
IF (C.LT.0.)C = 0.
B = SQRT(C)
```

En el caso $C < 0$ se cumple el operador de atribución $C = 0$, que no funciona en el caso $C \geqslant 0$.

Ejercicio 34. En caso de $x > 3$ calcular $A = -B + B\sqrt{x - 3}$. En los casos restantes suponer que $A = -B$.

Ejemplo. Cambiar de lugar los valores de las variables A y B, si $A > B$; en caso contrario dejar todo sin cambios.

```
      IF(A.LE.B) GO TO 1
      C = A
      A = B
      B = C
    1 CONTINUE
```

El primer operador comprueba la condición $A \leqslant B$, y si ésta se cumple (es decir, la expresión A.LE.B es real), entonces transmite el control al operador con la marca 1. Si, por el contrario, A > B, entonces el operador GO TO 1 no se ejecuta y se efectúa la transposición de las variables A y B. Esta transposición se realiza mediante tres operadores de atribución. Primeramente es necesario enviar el valor de la variable A a la célula C, y solamente después se puede inscribir en A el valor de B. Si no se indica el operador C = A, entonces el operador A = B borrará el valor anterior de la variable A. Después de la ejecución del operador A = B a la célula B se envía el valor de la variable A, conservado en la célula C.

2.1.9. Colecciones bidimensionales

Nosotros ya conocemos las colecciones unidimensionales. Cada elemento de semejante colección tiene un índice, por ejemplo A(I). En el Fortran se permiten también colecciones bidimensionales y tridimensionales. La multitud de asientos en el cine es un análogo de la colección bidimensional. Cada asiento se determina por dos

coordenadas: «fila» y «asiento». Las coordenadas que determinan la posición de cada elemento de la colección bidimensional son los índices: el número del renglón y el número de la columna.

Ejemplo. Supongamos que se ha dado la colección bidimensional C:

0.1	5.3	6.8	10.15
13.1	8.6	3.5	15.2
7.5	3.8	8.4	0.8

Esta colección contiene 3 renglones con 4 elementos en cada uno de ellos. El elemento de la colección se designa con el nombre C con dos índices (el número del renglón y el número de la columna), que se señalan entre paréntesis y se separan con comas. Por ejemplo, 8.6 es el elemento C (2,2), 15.2 es C (2, 4), 3.8 es C (3, 2).

Toda colección bidimensional se describe con un operador del tipo DIMENSION A (m, n). Aquí A es el nombre de la colección, m, el número de renglones, y n, el número de columnas. Los valores de m y n se dan en forma de constantes enteras sin signo. Por ejemplo, la colección C antes examinada se describe con el operador DIMENSION C (3, 4).

Ejemplo. Elevar al cuadrado todos los elementos del 3er renglón de la colección A que tiene 5 renglones y 7 columnas.

```
      DIMENSION A (5, 7)
      DO 2 J = 1,7
      A(3, J) = A(3, J)**2
    2 CONTINUE
```

Aquí la variable J que determina el número de la columna y del elemento de la colección A recorre todos los valores desde 1 hasta 7, mientras que el número del renglón permanece constante, igual a 3.

Ejemplo. Elevar al cuadrado todos los elementos de la colección A del ejemplo anterior.

```
  DO 2 I = 1, 5
  DO 2 J = 1, 7
2 A(I, J) = A(I, J)**2
```

Aquí se han utilizado dos ciclos que terminan con un mismo operador. El ciclo exterior realiza la selección de los números de los renglones desde 1 hasta 5. En el ciclo interior, para cada valor del número del renglón, se seleccionan todos los números de las columnas desde 1 hasta 7.

Ejemplo. Introducir en todos los elementos de la diagonal principal de la colección X(10, 10) el número 3. La diagonal principal va desde el elemento X(1, 1) hasta el elemento X(10, 10), pasando por los elementos X(2, 2), X(3, 3), etc.

```
  DIMENSION X(10, 10)
  DO 5 K = 1, 10
5 X(K, K) = 3.
```

Aquí los elementos de la diagonal principal tienen los índices (K, K), K = 1, . . ., 10.

2.1.10. Extracción de una colección bidimensional a la impresora

Comenzaremos por un ejemplo. Supongamos que la colección A se compone de 3 renglones con cuatro elementos en cada uno de ellos:

3.	2.1	8.	4.3
0.2	0.3	0.8	0.4
0.05	0.001	0.03	0.15

En la memoria de la máquina estos números se almacenan por columnas, es decir, en la sucesión siguiente: 3., 0.2, 0.05, 2.1, 0.3, 0.001, 8., 0.8, 0.03, 4.3, 0.4, 0.15. Para la extracción de todos los elementos de esta colección en aquella sucesión en la que éstos se encuentran almacenados en la memoria de la máquina es suficiente escribir el operador PRINT 42, A, donde 42 es la marca del operador FORMAT, A, el nombre de la colección. Por ejemplo, los operadores

```
    PRINT 42, A
42  FORMAT (10X, 12F6.3)
```

imprimirán todos los 12 números en un renglón.

Para la extracción de los elementos de la colección en otra sucesión se puede utilizar el ciclo implícito. Por ejemplo, si queremos extraer estos elementos por renglones, entonces se puede escribir el operador

PRINT 42, ((A(I, J), J = 1, 4), I = =1, 3).

Este operador actúa de la manera siguiente. Primero I adquiere el valor 1 y se

seleccionan todos los elementos del primer renglón con los números de columnas 1, 2, 3, 4 (J = 1, 4). Luego I adquiere el valor 2 y se repite la selección de los valores de J. El último valor de I será 3, con el que también se efectuará la selección de todos los valores de J desde 1 hasta 4. Si damos el operador FORMAT de tal forma que en cada renglón se impriman 4 números, entonces obtendremos en el papel todos los elementos de la colección en su «disposición natural», es decir, en forma de una tabla bidimensional que se compone de 3 renglones con 4 elementos en cada uno de ellos.

Los operadores

```
      PRINT 42, ((A(I, J), J = 1, 4),
     *I = 1, 3)
   42 FORMAT (5X, 4F6.3)
```

precisamente permiten obtener semejante disposición de los elementos de la colección A durante la impresión.

En el ciclo implícito se puede no seleccionar todos los valores de los renglones y de las columnas. Por ejemplo, si queremos extraer a la impresora solamente el 1er y 3er renglones de la colección A el PRINT correspondiente debe escribirse así:

```
PRINT 42, ((A (I, J), J=1, 4) I=1, 3, 2).
```

En caso general, el operador PRINT que contiene ciclo implícito puede escribirse:

```
   PRINT p, ((A (I, J), J=m1, n1, k1),
             I=m2, n2, k2).
```

```
PRINT p, ((A(I, J), J = m1, n1, k1),
I = m2, n2, k2).
```

Aquí p es la marca del operador FORMAT, m1 y m2, los valores iniciales de los índices J e I, n1 y n2 son los valores finales de estos índices. k1 y k2 son los valores de los pasos por los índices J e I. Los valores de las magnitudes m1, n1, k1, m2, n2, k2 deben ser estrictamente positivos.

El ciclo implícito se puede utilizar también en el operador READ. Por ejemplo, los operadores

```
    DIMENSION R (8, 6)
    READ 5, ((R (K, L), L = 2, 6, 2),
   *K = 1, 8, 5)
  5 FORMAT (6F8.2)
```

realizan la introducción de 6 números y la inscripción de éstos en los elementos R (1, 2), R(1, 4), R(1, 6), R(6, 2), R(6, 4), R (6,6).

2.1.11. Subrutina (subprograma)

Durante la solución de distintos problemas frecuentemente surge la necesidad de calcular con unos mismos algoritmos, por ejemplo, la raíz de la ecuación $f(x) = 0$ o hallar el elemento máximo en una colección bidimensional. ¿Será posible no programar semejantes algoritmos de nuevo cada vez, sino hacerlo una sola vez y utilizar los programas ya preparados?

En el lenguaje Fortran se prevé la posibilidad de reunir cualquier sucesión de operadores en una subrutina o subprogra-

ma particular. Para poner esta subrutina en marcha es necesario escribir los operadores correspondientes que la conectan al programa principal. En lo sucesivo precisaremos las nociones «programa» y «subprograma».

El subprograma o subrutina tiene el título del aspecto

SUBROUTINE S o

SUBROUTINE S (X, Y, Z, . . .).

Aquí S es el nombre de la subrutina, y X, Y, Z, . . ., los parámetros formales. Los estudiaremos más detalladamente.

Los *parámetros formales* son las denominaciones de las variables y de las colecciones a través de las cuales se transmite la información del programa principal al subprograma (datos iniciales) o del subprograma al programa principal (resultados).

Supongamos, por ejemplo, que hemos compuesto el subprograma de la solución de la ecuación cuadrática $ax^2 + bx + c = 0$:

SUBROUTINE SQ (A, B, C, X1, X2)

Éste contiene 5 parámetros formales, siendo así que los tres primeros se destinan para los coeficientes a, b y c, y los dos últimos se destinan para las raíces x_1 y x_2 de la ecuación. La definición «formales» subraya que el subprograma resuelve el problema de forma general.

El autómata para la venta de billetes para los trenes suburbanos puede servir de ilustración del principio de trabajo de un

subprograma. Este autómata está puesto punto para funcionar con distintos juegos de monedas (parámetros formales) y despacha billetes hasta diferentes estaciones de destino (resultado). Para que el autómata funcione es necesario echar monedas de valor concreto (parámetros reales).

Para poner en marcha un subprograma es menester tener acceso a él (o llamarle). La llamada del subprograma (o el acceso al subprograma) se efectúa con el operador CALL. En este operador se indican el nombre del subprograma y, entre paréntesis, los parámetros reales (si el subprograma tiene parámetros formales). Por ejemplo, para la llamada del subprograma SQ (A, B, C, X1, X2) debemos señalar en el operador CALL los valores de los cinco parámetros reales que corresponden a A, B, C, X1 y X2. Si, por supuesto, queremos resolver la ecuación $5x^2 - 12x + 3 = 0$ y designamos sus raíces por T y C, entonces la llamada del subprograma SQ será la siguiente:

CALL SQ(5., −12., 3., T, C)

No hay necesidad de dar los valores de los coeficientes de la ecuación en forma de constantes.

Si, por ejemplo, el valor del coeficiente superior de la ecuación está incluido en la variable P, y los valores de los coeficientes restantes están incluidos, respectivamente, en Q y R, entonces el acceso a SQ puede ser el siguiente:

CALL SQ (P, Q, R, T, C)

El programa en el que se indica el operador CALL se denomina *programa de llamada*.

Durante la ejecución del operador CALL se efectúan las siguientes operaciones. La información respecto a los parámetros reales (si éstos existen) se envía a n células de la memoria dispuestas consecutivamente, donde n es el número de parámetros. Luego el control se transmite al comienzo del subprograma que se llama.

El subprograma recibe, a través de las células precitadas, la información respecto a los parámetros reales y establece una correspondencia biunívoca entre los parámetros formales y reales. A continuación se ejecuta el cumplimiento del subprograma.

Una vez ejecutado el subprograma, el control se transmitirá en el programa de llamada al operador que sigue al operador CALL.

Compongamos el subprograma SQ (A, B, C, X1, X2) para la solución de una ecuación cuadrática con coeficientes A, B, C que, en calidad de resultados, da los valores X1 y X2 de las raíces reales. Vamos a suponer que con los valores dados de los coeficientes la ecuación tiene raíces reales.

```
SUBROUTINE SQ (A, B, C, X1, X2)
D = SQRT (B**2-4.*A*C)
X1 = (-B + D)/(2.*A)
X2 = (-B - D)/(2.*A)
RETURN
END
```

En el subprograma SQ encontramos el operador desconocido RETURN. Este operador efectúa el regreso (es decir, la transferencia de control) al programa de llamada. Este operador no es idéntico al operador END que da a «conocer» al compilador que ya no hay más operadores que tengan relación con el subprograma dado. En el subprograma pueden haber varios operadores RETURN, mientras que el operador END siempre es uno solo.

El subprograma SQ que compusimos permite resolver ecuaciones cuadráticas con discriminante no negativo. Pero, ¿qué ocurrirá si en el programa de llamada se dan tales valores de los coeficientes que el discriminante que se obtiene es negativo (por ejemplo, CALL SQ (3., 1., 7.5, RT, PSI)? En este caso tendrá lugar una *parada por avería* (fin anormal), es decir, la interrupción del trabajo del programa al intentar extraer la raíz de un número negativo.

Una situación análoga puede surgir cuando cualquiera de los parámetros reales se indique en forma de constante o variable entera (y no real). Por ejemplo, durante el acceso

CALL SQ (3, 1., 7.5, RT, PSI)

donde en lugar de la constante real 3. se indica la constante entera 3, el subprograma SQ puede no funcionar. Lo mismo puede suceder también cuando se indican más o menos parámetros reales de los que se deben indicar.

Así, el subprograma exige el uso correcto, en caso contrario este subprograma puede dar un resultado incorrecto o su trabajo puede interrumpirse.

De manera análoga se porta el autómata para la venta de billetes antes mencionado. El intento de utilizar monedas «malas» (encorvadas, de otro valor) puede conducir o al hecho de que no despache el billete o a que se rompa el autómata.

Ejemplo. Examinemos cómo se puede hacer uso del subprograma SQ para hallar la suma de los cubos de las raíces de las ecuaciones

$$3x^2 + 8x - 0{,}1 = 0 \text{ y } -6y^2 + 12y + 0{,}3 = 0.$$

```
  PROGRAM RESULT
  CALL SQ (3., 8.,          Para la solución de
 *-0,1, X1, X2)             la 1ª ecuación
  CALL SQ (-6., 12.,        Para la solución de
 *0.3, Y1, Y2)              la 2ª ecuación
  S=X1**3 + X2**3 +
 *Y1**3 + Y2**3
  PRINT 5, S
5 FORMAT (6X,
 *5HSUMA=, E12.3)
  END
```

Después de la ejecución de dos operadores CALL las variables X1, X2, Y1, Y2 contendrán los valores de las raíces de las ecuaciones dadas.

2.1.12. Subrutina-función

Otro procedimiento de presentación del algoritmo en el Fortran es la subrutina-función (subprograma-función). Su encabezamiento tiene el aspecto

FUNCTION F (X, Y, Z, . . .)

En el subprograma-función, a diferencia de la SUBROUTINE, debe existir obligatoriamente por lo menos un parámetro formal. Otra diferencia entre FUNCTION y SUBROUTINE, más importante, consiste en que estos subprogramas se llaman del programa principal de manera distinta. Para extraer el subprograma-función es necesario indicar su nombre conjuntamente con la lista de los parámetros reales en cualquier expresión aritmética. Supongamos, por ejemplo, que el subprograma-función AMAX (X, Y) calcula el valor del mayor de los parámetros X e Y. Entonces el operador R = AMAX (T, P*Q) significa la atribución por la variable R del mayor de los valores de las dos magnitudes T y P*Q.

Una particularidad distintiva de FUNCTION es también el hecho de que el resultado de su trabajo es un número, al que se le atribuye el nombre de esta FUNCTION. Por ejemplo, el subprograma-función AMAX ya mencionado puede ser presentado así:

```
FUNCTION AMAX (X, Y)
AMAX = X
```

```
IF (X.LT.Y) AMAX = Y
RETURN
END
```

El nombre del subprograma-función determina el tipo del resultado que éste da. Por ejemplo, el subprograma-función estándar, que calcula el logaritmo natural, se denomina ALOG (X), y no LOG (X). En el último caso el nombre comienza por la letra L y determina una magnitud de tipo «entero».

El subprograma-función y el subprograma, a pesar de su diferencia, tienen mucho de común. En uno y otro los parámetros reales pueden ser expresiones aritméticas. El tipo y orden de seguir de los parámetros reales debe corresponder estrictamente al tipo y orden de los parámetros formales.

Si entre los parámetros formales hay una colección ésta debe estar descrita por el operador DIMENSION. Hasta el momento tuvimos que ver con colecciones cuya dimensión se daba en forma de constantes enteras, por ejemplo DIMENSION X(50). En los subprogramas y subprogramas-funciones en calidad de parámetros formales se pueden indicar colecciones, cuyas dimensiones se dan en forma de variables enteras, por ejemplo, DIMENSION R(N). En este caso los nombres R y N deben estar indicados en la lista de los parámetros formales.

Ilustraremos lo dicho con un ejemplo. Compongamos un subprograma-función que

calcule el valor máximo del elemento de la colección A, si A consta de N elementos.

```
      FUNCTION RMAX (A, N)
      DIMENSION A (N)
      RM = A (1)
      DO 1 I = 2, N
    1 IF (A(I).GT.RM) RM = A (I)
      RMAX = RM
      RETURN
      END
```

Aquí, en calidad de A, se puede dar cualquier colección que contenga no menos de dos elementos. A la salida (como resultado) se obtiene el valor máximo entre los elementos de la colección dada. Prestemos atención al último operador del ciclo DO— IF lógico. Semejante terminación del ciclo es admisible. Con esto, en caso de la autenticidad de la expresión indicada entre paréntesis, se efectúa el operador RM = = A(I), y en caso de la falsedad de la expresión, la continuación del ciclo.

Se debe mencionar particularmente que el subprograma y el subprograma-función obtienen información mediante los parámetros formales y respecto al programa de llamada no «saben» nada más (véase también 2.1.13). De manera semejante el autómata que despacha billetes también reacciona solamente a las monedas que echaron en él, y no «sabe» nada sobre aquellas que usted tiene en el bolsillo.

En forma de subprogramas-funciones se hace el cálculo de las funciones elementales: sen x, ln x, e^x y otras. Puesto que los valores de estas funciones frecuentemente se utilizan en los programas, los subprogramas-funciones correspondientes se formalizan como una biblioteca de subprogramas-funciones estándares. Estas funciones se calculan con exactitud que se aproxima a la máxima posible para el ordenador dado y lo más rápido posible. Aduciremos datos sobre los subprogramas-funciones estándares (o de biblioteca).

En la tabla (véase la pág. 118) se utilizan las designaciones siguientes:

X, es una variable real,
CX, variable compleja,
DX, con exactitud doble,
IX, entera.

Por la primera letra del nombre de estos subprogramas se puede determinar el tipo del resultado: si la primera letra es I, J, K, L, M, N, entonces el tipo es «entero», si es la letra C, entonces es «complejo», si es D, el tipo es «con exactitud doble», y en todos los demás casos el tipo del resultado es «real».

2.1.13. El operador COMMON

El programa que llama cualquier subprograma o subprograma-función intercambia información con éstos mediante los pará-

Destino	Acceso	Tipo del resultado
Exponente		
e^{X}	EXP(X)	Real
e^{CX}	CEXP(CX)	Complejo
e^{DX}	DEXP(DX)	Con exactitud doble
Logaritmo natural		
ln X	ALOG(X)	Real
ln CX	CLOG(CX)	Complejo
ln DX	DLOG(DX)	Con exactitud doble
Logaritmo decimal		
log X	ALOG10(X)	Real
log DX	DLOG10(DX)	Con exactitud doble
log CX	CLOG10(CX)	Complejo
Arco tangente		
arctg X	ATAN(X)	Real
arctg DX	DATAN(DX)	Con exactitud doble

arctg CX	CATAN(CX)	Complejo
Raíz cuadrada		
$\sqrt{X}$	SQRT(X)	Real
$\sqrt{CX}$	CSQRT(CX)	Complejo
$\sqrt{DX}$	DSQRT(DX)	Con exactitud doble
Seno de un ángulo (en radianes)		
sen X	SIN(X)	Real
sen CX	CSIN(CX)	Complejo
sen DX	DSIN(DX)	Con exactitud doble
Coseno de un ángulo (en radianes)		
cos X	COS(X)	Real
cos CX	CCOS(CX)	Complejo
cos DX	DCOS(DX)	Con exactitud doble
Módulo de un número		
$\lvert X \rvert$	ABS(X)	Real
$\lvert IX \rvert$	IABS(IX)	Entero
$\lvert CX \rvert$	CABS(CX)	Real
$\lvert DX \rvert$	DABS(DX)	Con exactitud doble

Continuación de la tabla

Destino	Acceso	Tipo del resultado
Parte entera del número		
[X]	AINT(X)	Real
[X]	INT(X)	Entero
[DX]	IDINT(DX)	Entero
[DX[	DINT(DX)	Con exactitud doble
Resto de la división		
X/Y	AMOD(X, Y)	Real
IX/IY	MOD(IX, IY)	Entero
Elección del elemento máximo entre		
X1, X2, X3, ... XN	AMAX1(X1, X2, ...)	Real
IX1, IX2, IX3, ... IXN	AMAX0(IX1, IX2, ...)	Real
IX1, IX2, IX3, ... IXN	MAX0(IX1, IX2, ...)	Entero
X1, X2, X3, ... XN	MAX1(X1, X2, ...)	Entero
DX1, DX2, DX3, ... DXN	DMAX1(DX1, DX2, ...)	Con exactitud doble
Elección del elemento mínimo entre		
X1, X2, X3, ... XN	AMIN1(X1, X2, ...)	Real

IX1, IX2, IX3, ... IXN	AMIN0(IX1, IX2, ...)	Real
IX1, IX2, IX3, ... IXN	MIN0(IX1, IX2, ...)	Entero
X1, X2, X3, ... XN	MIN1(X1, X2, ...)	Entero
DX1, DX2, DX3, ... DXN	DMIN1(DX1, DX2, ...)	Con exactitud doble
Formación de la parte principal del argumento con exactitud doble	SNGL(DX)	Real
Formación de la parte real de la magnitud compleja CX	REAL(CX)	Real
Formación de la parte imaginaria del número complejo CX	AIMAG(CX)	Real
Formación del tipo "con exactitud doble"	DBLE(X)	Con exactitud doble
Asociación de X1 y X2 en una magnitud compleja $X1+iX2$	CMPLX(X1, X2)	Complejo
Obtención del número complejo conjugado del CX dado	CONJG(CX)	Complejo
Tangente del ángulo (en radianes) tg X	TAN(X)	Real

metros. En el Fortran existe también otro procedimiento de transmisión de la información: a través de las colecciones comunes de células, denominadas COMMON-*bloques* o *bloques comunes.*

Para la descripción de semejantes bloques sirve el operador declarativo COMMON. Este operador tiene el aspecto

COMMON /A/B, C, D, . . ., X

Aquí A es el nombre del bloque común, B, C, D, . . ., X son variables que entran en este bloque.

Supongamos que en PROGRAM MIST se indica el operador

COMMON /LXTP/R(15), C, M28A

donde C y M28A son variables simples. Esto significa que en la memoria se ha separado un bloque de 17 células que puede utilizar cualquier subprograma (o subprograma-función).

Sea que deseamos que SUBROUTINE IVAR obtenga acceso a este COMMON-bloque. Entonces en el subprograma se debe describir el bloque con el nombre LXTP con longitud de 17 células.

El nombre del bloque debe ser el mismo en los programas de llamada y que se llama. Los nombres de las variables y de las colecciones que entran en el bloque se eligen a juicio del programador y pueden no coincidir con los nombres elegidos en PROGRAM. Supongamos que en SUBROUTINE IVAR

se indica el operador

```
COMMON /LXTP/A(8), B(8), NT
```

La primera célula del bloque se llama R(1) en MIST, mientras que en IVAR se llama A(1). Así, R(1) y A(1) son las designaciones de una misma célula en distintos programas.

Si el subprograma utiliza COMMON-bloque dicho subprograma puede no tener parámetros.

Ejemplo. Ilustremos esto en el ejemplo del subprograma de solución de la ecuación cuadrática que ya fue escrito en la variante que tiene 5 parámetros formales. En la nueva variante en lugar de los parámetros se utilizará el COMMON-bloque.

```
SUBROUTINE SQCOM
COMMON/AB/ A, B, C, X1, X2
D = SQRT (B**2—4.*A*C)
X1 = (—B + D)/(2.*A)
X2 = (—B — D)/(2.*A)
RETURN
END
```

Para poder utilizar semejante subprograma es necesario describir el bloque AB en el programa de llamada. Antes de dirigirse a SQCOM se deben inscribir en las primeras 3 células de COMMON-bloque los valores de los coeficientes de la ecuación. Después del acceso a SQCOM en la 4ª y en la 5ª células de COMMON-bloque se en-

contrarán los valores de las raíces de la ecuación.

En general, un mismo algoritmo se puede presentar en forma de distintos subprogramas que se diferencian por el número de parámetros formales.

Durante el acceso al subprograma el número de parámetros reales debe ser exactamente el mismo que el de los parámetros formales. Si incluso cualquier parámetro real no es vital para la solución de un problema concreto, durante el acceso al subprograma (o al subprograma-función) debe ser indicado. En este sentido los COMMON-bloques permiten una gran libertad en su utilización.

Si la información que contiene el COMMON-bloque no se necesita, en el programa se puede no describir el COMMON-bloque correspondiente. Por esto, al componer el subprograma, generalmente se sigue el principio siguiente. La información necesaria para el funcionamiento del subprograma, y asimismo los resultados fundamentales, se presentan en forma de parámetros formales. A veces también representan interés los valores intermedios de algunas variables del subprograma. Semejantes resultados se dan a través del COMMON-bloque.

Para la solución del problema en el ordenador electrónico podemos elaborar o bien solamente un PROGRAM (programa director) o bien PROGRAM y varias SUBROUTINE y FUNCTION. En cual-

quier caso obligatoriamente debe haber PROGRAM (además, solamente uno), mientras que la cantidad de FUNCTION y SUBROUTINE puede ser arbitraria. El programa (PROGRAM) y los subprogramas (FUNCTION, SUBROUTINE), que entran en el paquete de cualquier problema, se denominan *módulos del programa.*

Prestemos atención a una circunstancia más ligada a los nombres de las variables (o de las colecciones) que se utilizan en los subprogramas. *Si el nombre no es ni parámetro formal, ni parámetro real, ni elemento del COMMON-bloque, entonces los otros módulos del programa no tienen acceso a la célula (células) correspondiente de la memoria.* Por ejemplo, en el subprograma SQCOM semejante nombre es D. Este nombre es local, es decir, «propiedad privada» del subprograma dado. Si es necesario la magnitud D puede hacerse accesible para los otros módulos del subprograma. Por ejemplo, se puede describir la variable D en cierto COMMON-bloque. No obstante, semejante posibilidad debe ser inculcada durante la escritura del subprograma SQCOM.

2.1.14. Estructura modular de los programas

Hasta ahora tratamos de subprogramas aislados (módulos) del programa Fortran. Ahora estudiaremos el programa en conjunto. El trabajo comienza por el programa

director (por el módulo director) que tiene el título PROGRAM. Cualquier otro módulo (FUNCTION o SUBROUTINE) puede ser llamado del PROGRAM. Sin embargo, ningún módulo puede llamar el programa director.

Ejemplo.

```
  PROGRAM P
  A = 15.5
  B = SIN(A)
  PRINT 2, B
2 FORMAT (3X, 'B =', F8.6)
  END
```

Aquí trabajan dos módulos: P y SIN(X). SIN(X) se llama de P. Esquemáticamente:

P → SIN(X).

La llamada de los módulos puede ser multigradual, es decir, si P, S1, S2, S3, S4 son los nombres de los módulos entonces puede haber, por ejemplo, una llamada del aspecto siguiente

```
P  →  S1
↓     ↓
S2 →  S3 → S4
```

es decir, el subprograma P llama los módulos S1 y S2. Cada uno de estos módulos llama el módulo S3 que, a su vez, llama a S4.

Pero durante la llamada de los módulos es inadmisible la recurrencia, es decir, los módulos posteriores en el esquema de lla-

madas no puedon recurrir a los anteriores. Por ejemplo, no puede existir el siguiente esquema:

P → S1 → S2 → S3 → S1

La estructura modular de los programas no permite poner a punto directamente todo el programa, sino cada módulo por separado.

Ejemplo. Supongamos que nuestro programa se compone de tres módulos: P, S1(T) y F(X).

```
PROGRAM P
. . .
CALL S1(T)
. . .
END

SUBROUTINE S1(T)
. . .
A = F(X)
. . .
RETURN
END
FUNCTION F(X)
. . .
RETURN
END
```

El esquema de llamada de los módulos es el siguiente:

P → S1 → F

¿Cómo poner a punto este programa si los módulos P, S1 y F son bastante complicados? Existen dos métodos de corrección: *descendente* y *ascendente*.

1. El método descendente de puesta a punto se utiliza al crear complejos programáticos grandes. Primero se corrige el PROGRAM. Con esto, en lugar de todos los módulos restantes, se insertan «tapones», es decir, solamente se dejan los títulos, se escriben los operadores de atribución necesarios, luego RETURN y END.

Por ejemplo,

```
SUBROUTINE S1(T)
T = 0.5
RETURN
END
FUNCTION F(X)
F = 2.
RETURN
END
```

Ahora se puede poner a punto PROGRAM, basándose en los resultados de trabajo de los módulos restantes, conocidos de antemano.

Una vez ajustado PROGRAM, empieza el trabajo de ajuste con S1(T). Se quita el «tapón» S1 y se coloca en el programa el módulo real S1, y en lugar de F se pone el «tapón».

```
FUNCTION F(X)
F = 2.
```

```
RETURN
END
```

Una vez puesto a punto el módulo S1, en lugar del «tapón», se inserta el texto original FUNCTION F(X) y se pone a punto este nivel, el más bajo.

Con ello, en cada etapa, se tiene que ver con un solo módulo (los demás o ya están corregidos o están sustituidos por «tapones»). Frecuentemente, además, durante el proceso de puesta a punto se revelan exigencias complementarias respecto a los módulos restantes. Por ejemplo, a menudo los módulos del nivel inferior se escriben solamente después de haber corregido los precedentes.

2. En el método ascendente de puesta a punto se comienza por el nivel inferior. En nuestro ejemplo, éste es la corrección de FUNCTION F(X). Para ello se escribe un PROGRAM de simpleza máxima que tenga acceso a F(X) y que imprima el resultado.

Después de que el programador se convenza de la justeza del funcionamiento de F(X) comenzará a poner a punto S1. Para ello el programador escribe un PROGRAM nuevo, que solamente llama a S1 e imprime su resultado, pero en S1 ya funciona F puesto a punto. Así, el nivel superior se corrige por el nivel inferior ya puesto a punto.

La estructura modular del programa también permite encargar la elaboración de un programa complicado a varias personas. Cada uno de los participantes compone y po-

ne a punto uno o varios módulos formalizados en forma de FUNCTION o SUBROUTINE.

Los módulos que más frecuentemente se utilizan se reúnen en *bibliotecas de subprogramas*. Los módulos de biblioteca están previamente puestos a punto y constantemente se encuentran presentes en la memoria del ordenador. Si en el programa existe acceso a cualesquiera módulos de biblioteca éstos, durante la carga del programa en la memoria del ordenador, se «ponen en fila» con aquellos módulos que escribió el propio programador. De tal manera, durante el funcionamiento del programa, en un mismo sector de la memoria se encuentran todos los módulos que se utilizan en este programa.

2.2. ¿Cómo funciona el programa traductor?

El programa escrito en Fortran e introducido en la máquina debe ser transformado por el traductor (compilador) en una sucesión de instrucciones de máquina.

Examinemos en rasgos generales cómo transcurre el proceso de trabajo del compilador. El conocimiento de este proceso puede resultar útil tanto al escribir programas como durante su puesta a punto.

Estudiaremos el proceso de funcionamiento del compilador en un ejemplo concreto. Supongamos que existe el siguiente programa:

```
      PROGRAM SUMMA
      N = 100000
      S = 0.
      DO 1 K = 1, N
    1 S = S + 1./K
      PRINT 2, S
    2 FORMAT (1X, 2HS =, E12.6)
      END
```

El primer operador a cumplir (N = = 100000) contiene la variable N y la constante 100 000. El traductor inscribirá en una célula el valor de la constante, asignará otra célula para la variable N, guardará en la memoria los números de estas células y los utilizará al componer la instrucción de máquina, o de varias instrucciones (según sea el tipo del ordenador), que realizan el envío de 100 000 a la célula N.

El operador S = 0. se traduce de manera análoga a la anterior. Si en lugar de S = = 0. se escribió S = 0 (es decir, se señala el cero «entero») entonces, para el cumplimiento de este operador, se necesitarían más instrucciones de máquina que para el primero. El hecho consiste en que los tipos de la variable y de la constante en el caso S = 0 no coinciden. Por esto, antes de cumplir la operación del envío del cero, la máquina debe transformar el cero «entero» en cero «real». Para no alargar el programa de máquina se debe cuidar de la correspondencia de los tipos de constantes y variables.

En el operador DO 1 K = 1, N señala la marca del último operador del ciclo. Este operador aparecerá más tarde en el programa, y mientras tanto el compilador introducirá la marca 1 en una lista especial. El traducir los operadores siguientes se comprobará si coincide o no la marca del operador con la marca 1 de la lista. En caso de coincidencia se considera que el operador es el último en el ciclo. Puede resultar que en el programa no exista un operador con la marca 1. Entonces, al llegar a END, el compilador dará la diagnosis respecto a la ausencia del último operador del ciclo DO. En el operador que examinamos DO 1 K = 1, N se indica la constante 1 y las variables K y N. La constante 1 es enviada por el traductor a una célula de la memoria, para K se asigna su célula, y para N la célula ya fue asignada antes.

El operador siguiente 1 S = S + 1./K calcula el valor de S y simultáneamente es el último operador del ciclo. En este operador se tropieza con la nueva constante 1. (de otro tipo que 1), que el compilador envía a la célula destinada. Las variables S y K ya se encontraron antes, y en las instrucciones de máquina pueden utilizarse los correspondientes números de las células. Después de la traducción del operador 1 S = S + 1./K se formarán las instrucciones que comprueban la condición de la terminación del ciclo.

El operador PRINT, que se indica después del ciclo, se traducirá algo diferente

que los anteriores. Es que para la extracción de cada número a la impresora es necesario cumplir muchas instrucciones de máquina. Se requiere transformar cada número que se extrae de la interpretación de máquina en interpretación decimal, en concordancia con la especificación del formato. Se necesita también indicar las instrucciones que controlan el trabajo del dispositivo impresor. Por esto, para la extracción de los números a la impresora se utiliza un programa previamente elaborado que se encuentra en la máquina. El compilador solamente forma las instrucciones de acceso al programa señalado. Estas instrucciones determinan las células de las que se debe tomar la información para la salida a la impresora. En ellas se indica también dónde se encuentra el operador FORMAT que corresponde al operador dado PRINT.

El operador FORMAT, a diferencia de los anteriores, no es un operador a cumplir. Por esto a él no le corresponderán ningunas instrucciones de máquina. No obstante, para este operador se designará un sector determinado de la memoria. Los símbolos de los que se compone el operador FORMAT (es decir, todo lo que comienza por la abertura de paréntesis) se transformarán por el traductor en la presentación de máquina y se alojarán en una o varias células de la memoraia.

El compilador traslada todos los operadores FORMAT al fin del programa, disponiéndolos conjuntamente con las cons-

tantos y células destinadas para las variables y colecciones.

El operador END determina el fin del programa. Al encontrar este operador, el compilador efectúa la formalización del resultado de la traducción.

2.2.1. Errores revelados por el compilador

En el programa presentado para la traducción pueden haber errores. Aquellos que están relacionados con la infracción de las reglas del lenguaje Fortran serán revelados por el traductor.

En cada traductor se prevé la entrega de comunicados (diagnosis) respecto a los errores cometidos en el programa. La diagnosis se puede dar o bien inmediatamente después de la impresión del texto de cada operador erróneo, o bien después de la traducción de todo el programa, a la vez para todos los operadores erróneos. En el último caso cada texto diagnóstico va acompañado del número del renglón del programa al que pertenece.

El contenido del texto de la diagnosis puede o bien indicar el carácter del error (por ejemplo, «paréntesis no cerrados»), o bien determinar sólo el número del error, por el que en un vademécum especial se puede encontrar el texto correspondiente.

Muchos programadores inexpertos a veces presentan quejas al traductor con motivo del desacuerdo del texto de la diagnosis con el carácter del error cometido. Semejan-

te desacuerdo se explica por el hecho de que el traductor reacciona a la consecuencia del error, y no a la causa de ésta. Supongamos por ejemplo, que al escribir el operador A = B + 1.5E3. omitimos el signo +. Como resultado se obtendrá el operador de atribución A = B1.5E3, en el segundo miembro del cual se indica una expresión que contiene el símbolo inadmisible de «punto decimal». En la diagnosis se señalará precisamente que ilegítimamente se ha puesto un punto decimal.

También es necesario tener en cuenta que muchos operadores del programa están correlacionados. Por ejemplo, si en el primer miembro del operador de atribución se indica una variable con índices entonces en el programa se debe describir la colección correspondiente. Si la descripción de la colección está ausente o está hecha erróneamente, entonces la utilización de la variable con índices se convierte en ilegítima y será acompañada de una diagnosis. Semejante diagnosis se denomina *sugerida*.

Generalmente el traductor da la diagnosis por el primer error revelado en el operador. Con esto, el operador erróneo se excluye del programa, lo que puede provocar una diagnosis sugerida por otros operadores.

Examinemos ahora varios ejemplos.

```
SUBROUTINE FC
DIMENTION A (20)
A (1) = 5.3
DO 5 I = 2, 20
```

```
 5 A (I) = A (I — 1)**2 + 1.
   PRINT 23, A
23 FORMAT (10 (1X, F10.5))
   END
```

Aquí el error se cometió al describir la colección A: en el operador DIMENSION en lugar de la letra S se señaló la letra T. Por esto el operador DIMENTION A (20) se reconocerá como no identificado. Como resultado parecerá no existir la descripción de la colección A. Esto traerá consigo la presentación de la diagnosis por el operador A (1) = 5.3 y por el operador con la marca 5. De este modo, como resultado del único error, aparecerá el comunicado del traductor respecto a tres errores.

A veces, como resultado del error cometido, puede variar el sentido del operador. En este caso el traductor ya no puede notar nada, pues no tiene lugar la infracción de las reglas del Fortran. He aquí un ejemplo de semejante género de error.

```
   PROGRAM SEQUEN
   A = 1.
   DO 7 I = 1.20
 7 A = A + 1./A**2
   PRINT 36, A
36 FORMAT (1X, E12.6)
   END
```

El resultado dado por el programa es igual a 2. Esto testimonia el hecho de que

el ciclo fue realizado, juzgando por todo, una sola vez, y no 20 veces como propuso hacer el autor del programa. El error se cometió en el operador DO, en el que en lugar de coma se indicó un punto. ¿Por qué el compilador no notó este error? Simplemente porque ahora en lugar del operador DO se obtuvo un operador de atribución, en el primer miembro del cual se señaló la variable DO 7 I, y en el segundo miembro la constante 1.20. Señalemos que el traductor no toma en consideración los blancos indicados en la escritura del operador.

Tocaremos ahora otros errores que el traductor no puede revelar. Sea, por ejemplo,

```
      PROGRAM DIMEN
      DIMENSION A (100)
      DO 3 I = 1, 100
    3 A (I + 1) = 0.
      PRINT 16, A
   16 RORMAT (2X, 20F5.0)
      END
```

Aquí, cuando I = 100 tendrá lugar la inscripción del cero en el elemento A (101). Pero la colección A solamente contiene 100 elementos. Como resultado se «deteriorará» la célula de la memoria que sigue a aquella que corresponde al elemento A (100). Las consecuencias de este error pueden ser muy diferentes: todo depende del papel que la célula «deteriorada» desempeñe en el programa.

¿Puede revelar el compilador este error o no? Como regla no puede, pues el valor del índice se determina en el proceso del cálculo. No obstante, en ciertos traductores se prevé un régimen especial de trabajo, cuando en el programa traductor se insertan instrucciones complementarias. Éstas (en el proceso del cálculo) comprueban si el valor del índice se encuentra en los límites permisibles. Al aparecer un valor inadmisible del índice se da la diagnosis (durante el cálculo). Sin embargo, semejante comprobación puede demorar considerablemente el trabajo del programa, por lo que el régimen señalado de traducción generalmente se emplea sólo durante la puesta a punto de los programas.

La autonomía de cada módulo del programa es una particularidad del lenguaje Fortran (PROGRAM, SUBROUTINE y FUNCTION). Cada uno de ellos se traduce independientemente de los demás. Por esto los errores relacionados con la interacción de los módulos no pueden ser revelados por el traductor. He aquí un ejemplo de semejante error.

```
      PROGRAM DUBNA
      C = 3.14
      CALL LCT (C, 2.)
      A = SQRT (2.)
      PRINT 12, A
   12 FORMAT (2X, F12.8)
      END
```

```
SUBROUTINE LCT (A, X)
X = 0.
IF (A.GT.0.) X = — 1./A
RETURN
END
```

El subprograma LCT en calidad de resultado da o bien cero (si $A \leqslant 0$), o bien $-1./A$ (si $A > 0$). Este resultado se inscribe en la célula de la memoria que corresponde al parámetro formal X. En PROGRAM, en calidad de célula para el resultado se indica aquella en la que se encuentra la constante 2. Por esto, después de la ejecución del operador CALL en esta célula ya habrá no 2., sino — 1./3.14. Así, en lugar de extraer la raíz del número 2. la extraeremos de un número negativo, lo que conducirá a la interrupción del cálculo.

En resumen señalaremos que la eliminación de los errores en el programa se logra con una minuciosa puesta a punto. Con esto es importante que durante la puesta a punto se comprueben todas las variantes de funcionamiento del programa.

2.3. Ejemplos aleccionadores

Aquí se examinarán ejemplos que, a primera vista, pueden parecer simples. No obstante, su simpleza es engañadora. Generalmente el programador principiante no capta las dificultades disimuladas, ocultas en semejantes problemas. Como resultado,

la solución más simple y evidente puede resultar ser no satisfactoria.

Problema 1. Calcular a_{10000}, si $a_1 = 1$ y

$$a_{n+1} = a_n + \frac{1}{a_n^2 + 1}, \quad n \geqslant 1.$$

Los programadores principiantes encargan una colección de longitud de 10000, cada elemento de la cual está destinado para almacenar el miembro correspondiente de la sucesión. Aproximadamente se obtiene la siguiente solución del problema:

```
   PROGRAM A 10000
   DIMENSION A (10000)
   A (1) = 1.
   DO 1 I = 2, 10000
 1 A (I) = A (I — 1) + 1./(A (I —1)
  ***2 + 1).
   PRINT 10, A (10000)
10 FORMAT (1X, 7HA10000 = , E12.6)
   END
```

La solución, claro está, es justa. Sin embargo, aquí la memoria del ordenador se utiliza muy irracionalmente. Efectivamente, en calidad de resultado es necesario solamente recibir un número. Este número se calcula con recurrencia, siendo así que en cada paso se efectúa el cálculo del término siguiente de la sucesión a través del anterior. Todos los términos restantes de la sucesión, que se calcularon en los pasos anteriores, ya no se necesitan. De aquí

se deduce que no hay necesidad de conservar y almacenar todos los términos de la sucesión. Es suficiente tener solamente aquel término de la sucesión que se necesita en el paso dado del cálculo.

Así, en lugar de una colección, es suficiente tener una variable que contenga el valor del miembro corriente de la sucesión. He aquí la nueva solución, más racional.

```
      PROGRAM ANEW
      A = 1.
      DO 2 I = 2, 10000
    2 A = A + 1./(A**2 + 1.)
      PRINT 20, A
   20 FORMAT (1X, 2HA =, E12.6)
      END
```

Problema 2. Calcular $\sqrt{x^2 + y^2}$.

A primera vista todo se resuelve simplemente:

```
      PROGRAM SQRTXY
      READ 3, X, Y
    3 FORMAT (2E10.3)
      SQ = SQRT (X**2 + Y**2)
      PRINT 6, SQ
    6 FORMAT (2X, E10.3)
      END
```

No obstante, semejante programa puede calcular SQ no para todos los valores admisibles de X o Y.

Transformemos la expresión $\sqrt{x^2+y^2}$ de la forma siguiente:

si $|x|>|y|$ entonces $\sqrt{x^2+y^2}=$

$$=|x|\sqrt{1+\left(\frac{y}{x}\right)^2};$$

si $|x|\leqslant|y|$ e $|y|\neq 0$ entonces

$$\sqrt{x^2+y^2}=|y|\sqrt{1+\left(\frac{x}{y}\right)^2};$$

si $|x|\leqslant|y|$ e $|y|=0$ entonces

$$\sqrt{x^2+y^2}=0.$$

Semejante solución permite obtener el resultado tanto cuando los valores de x e y son grandes como cuando son pequeños.

```
      PROGRAM SQNEW
      READ 13 X, Y
   13 FORMAT (2E10.3)
      ABSX = ABS(X)          Introducimos va-
                             riables nuevas
      ABSY = ABS(Y)          ABSX y ABSY
      IF (ABSX—ABSY)
     *2, 2, 1
    1 SQ=ABSX*SQRT           |x| √(1+(y/x)²)
     *(1.+(Y/X)**2)
      GO TO 10
    2 IF (ABSY) 11, 12, 11
   11 SQ=ABSY*SQRT           |y| √(1+(x/y)²)
     *(1.+(X/Y)**2)
      GO TO 10
   12 SQ=0.                  √(x²+y²) = 0
   10 PRINT 15, SQ
```

```
15 FORMAT (1X, 3HSQ=, E10.3)
   END
```

La nueva solución, claro está, es mucho más complicada. No obstante, permite obtener el resultado prácticamente para todos los valores admisibles del argumento.

Problema 3. Calcular la raíz de la ecuación $x^2 + 2000x + 1 = 0$ que sea menor por su módulo.

Para la solución de este problema, por supuesto, se puede utilizar la fórmula

$$x = -1000 + \sqrt{1000^2 - 1}.$$

Sin embargo, aquí está oculta la contrariedad siguiente. El valor

$$\sqrt{1000^2 - 1} \approx 999,9995,$$

es decir, será muy cercano a 1000. La diferencia 1000—999,9995, que es igual a 0,0005, tendrá de este modo una precisión en 6 signos menor que los números iniciales.

La salida de la situación dada consiste en excluir, transformando la fórmula «desgraciada», la resta de dos magnitudes cercanas. Al multiplicar y dividir x por el número $1000 + \sqrt{1000^2 - 1}$, obtendremos

$$x = \frac{-1}{1000 + \sqrt{1000^2 - 1}}.$$

En este caso ya no habrá pérdidas de precisión.

Los ejemplos expuestos se han elegido especialmente de tal manera que llamen

la atención del programador principiante sobre ciertas dificultades «ocultas» con las que se tropieza durante la realización práctica del programa.

2.4. Al principiante usuario de un terminal

Los primeros ordenadores se encontraban en posesión completa del programador. Este obtenía la máquina a su disposición y, sentándose ante el tablero de mando, seguía el curso del problema. Al buscar un error el programador frecuentemente utilizaba el régimen de un *tiempo*, es decir, la máquina se paraba después del cumplimiento de cada instrucción, y por las bombillas del tablero se podía ver qué números participaban en la operación y qué resultado se obtenía.

Muy pronto se hizo evidente la baja eficacia de semejante utilización del ordenador: mientras que el programador reflexionaba sobre su paso siguiente la máquina estaba parada.

Luego, para elevar la eficacia de su utilización, el programador fue sustituido por un operador en el tablero de mando.

Actualmente en muchas máquinas el trabajo transcurre de la manera siguiente. Los operadores del ordenador escriben en la cinta magnética de entrada (o en el fichero del disco) los paquetes de programas que los programadores dieron para el cálculo. Los paquetes, consecutivamente, se elaboran en el ordenador y los resultados se inscriben en la cinta magnética (fichero del disco) de salida. Luego la información se imprime (o se da a perforar).

Estos perfeccionamientos elevaron la eficacia de utilización de la máquina, pero el trabajo del programador se hizo menos intenso y perdió una parte de su atractividad. Si antes el proceso de puesta a punto del programa consistía, prácticamente, sólo un buscar y corregir el error, ahora este proceso es fundamentalmente la espera del despacho de la información por la máquina. Habiendo en-

contrado el error en el programa y corregido éste, el programador obtiene los resultados, frecuentemente, sólo al día siguiente. Semejante productividad del trabajo del programador, inadmisiblemente baja, resultó ser la causa de que en las manos de éste cayeran dispositivos que creaban la ilusión de que se trabajaba en el tablero de mando del ordenador: *tableros portátiles* o *terminales*. Éstos se conectan a la máquina y se instalan tanto a corta distancia del ordenador como a distancias considerables.

El trabajo del usuario en el terminal está organizado de distinta forma en los diferentes tipos de ordenadores electrónicos y diversos sistemas operacionales. Pero casi todos los tableros portátiles permiten introducir la información en el ordenador, leer la información de las cintas magnéticas (discos), crear el paquete del problema, poner en marcha el problema para el cálculo.

Los primeros terminales se diseñaron con el fin de utilizar las máquinas de escribir. Actualmente en calidad de base de los tableros portátiles sirven, fundamentalmente, distintos tipos de displays alfanuméricos, que son una pantalla de televisión con teclado con el que se puede introducir información.

En la URSS uno de los displays más difundidos es el Videotón 340. Este dispositivo es cómodo en la explotación y seguro en el trabajo. Este display tiene los parámetros siguientes:

— superficie útil de la pantalla — 200 × × 140 mm²,

— número de renglones — 16,

— número de símbolos en el renglón — 80,

— capacidad de la memoria intermedia — 1280 símbolos,

— formato de imagen — matriz de puntos con dimensión 5 × 7 en el sistema de exploración televisiva,

— juego de signos: 26 letras latinas, 10 cifras, 28 signos especiales, 31 letras rusas (excepto ё y ъ).

Posibilidades de redacción:

— borradura de la imagen en la pantalla,

— colocación de la marca en la 1ª posición del 1er renglón,

— cambio de renglón,

— desplazamiento de la marca en 4 direcciones,

— sustitución, intercalación, borradura del símbolo,

— sustitución, intercalación, borradura del renglón,

— subrayado,

— elevación del cuadro en un renglón al llenar el último renglón.

Régimen de trabajo:

— OFF LINE (autónomo),

— ON LINE (intercambio de símbolos con la máquina),

— SEND (transmisión de las colecciones de información a la máquina),

— PRINT (impresión desde la pantalla a la impresora digital automática del display).

Este display se fabrica en dos variantes fundamentales: una variante con alfabeto latino y otra, con alfabetos ruso y latino. Nos detendremos en describir la segunda variante.

Todos los símbolos están aplicados a las teclas. El display posee su propia memoria (amortiguadora o de buffer), cuyo contenido se visualiza en la pantalla. En la memoria del display puede acumularse y almacenarse la información durante el trabajo del programador.

Al apretar la tecla se elabora un código correspondiente, que o bien va a parar a la memoria del display, o controla el trabajo de éste, o se transmite directamente al ordenador.

Teclas de establecimiento del régimen de trabajo. El display puede trabajar en el régimen OFF LINE — *régimen autónomo*, en el que la información que se forma en el teclado va a parar a la memoria del display. Al mismo tiempo se bloquea el intercambio de información con el ordenador.

La apretadura de la tecla «ON LINE» bloquea la transmisión desde el teclado a la memoria del

display y permite el intercambio de información con el ordenador.

La tecla «SEND» sirve para que la información formada en el régimen OFF LINE, que se encuentra en la memoria del display (en la pantalla), se pase al ordenador. En este caso la comunicación con la máquina es unilateral: se permite la transmisión de información del display al ordenador, pero se bloquea la recepción.

Representemos esquemáticamente con una flecha las direcciones permitidas de transmisión de la información. Entonces los diferentes regímenes de trabajo del display aparecerán de la manera siguiente:

OFF LINE: teclado → memoria del display.

ON LINE: teclado → ordenador, ordenador → memoria del display.

SEND: memoria del display → ordenador electrónico.

PRINT: memoria del display → DIAN (dispositivo impresor alfanumérico) del display.

La tecla «OFF LINE» es poseedora de una prioridad máxima. Al apretar ésta el display pasa de cualquier régimen al régimen OFF LINE.

El display puede pasarse a los regímenes ON LINE y SEND solamente desde el régimen OFF LINE; al régimen PRINT se puede pasar tanto desde OFF LINE como de ON LINE. Desde el display se puede realizar la impresión transmitiendo el código correspondiente (véase la tabla de codificación) desde el ordenador. Una vez concluida la impresión el display regresa a aquel régimen desde el que entró en el régimen PRINT.

En las teclas alfanuméricas figuran dos símbolos: *superior* e *inferior*. Al apretar cualquier tecla se genera el símbolo inferior si la tecla «LAT» está apretada, y el símbolo superior si dicha tecla no se aprieta.

Por ejemplo, apretando solamente la tecla «Г/G» obtenemos el símbolo Г; y «hundiendo» la tecla «LAT», obtenemos la G.

Para la redacción del texto visualizado en la pantalla se emplean las *teclas de redacción:*

«ERASE» quita el contenido de la pantalla,

limpia la memoria amortiguadora, instala el trazo (cursor) en la primera posición del primer renglón;

«→», desplaza el trazo a una posición a la derecha (de la última posición del renglón a la primera posición del renglón siguiente, y de la última posición del último renglón a la primera posición del primer renglón);

«←», desplaza el trazo a una posición a la izquierda (de la primera posición del renglón a la última posición del renglón anterior; de la primera posición del primer renglón a la última posición del último renglón);

«↑», pasa el trazo hacia arriba en un renglón, y del primer renglón al último;

«↓», pasa el trazo hacia abajo en un renglón, y del último al primero;

«LINE FEED», pasa el trazo a la primera posición del renglón siguiente, y del último renglón a la primera posición del primer renglón;

«HOME», pone el trazo desde cualquier posición en la primera posición del primer renglón;

«TAB», pone el trazo en la primera posición del renglón siguiente, o después del símbolo |, si éste figura en el renglón dado:

«IC», se inserta un blanco en lugar del símbolo que señala el trazo; todo el renglón, comenzando desde este símbolo, se desplaza hacia la derecha en una posición (desaparece el símbolo de la última posición del renglón);

«DC», el símbolo señalado por el trazo desaparece, y la parte que queda a su derecha se desplaza hacia la izquierda en una posición (en la última posición del renglón aparece un blanco);

«IL», en lugar del renglón señalado por el trazo se forma un renglón de blancos, y todos los renglones, comenzando desde el dado, se desplazan hacia abajo en un renglón (el último renglón de la pantalla se pierde);

«DL», el renglón señalado por el trazo desaparece, y los renglones dispuestos por debajo suben en un renglón (el último renglón de la pantalla se llena de blancos);

Con ayuda de estas teclas se puede redactar el texto compuesto.

Para limpiar la pantalla y comenzar la composición de un texto nuevo es suficiente apretar la tecla «ERASE».

Apretando la tecla «ETX» se termina el texto compuesto para su transmisión en el régimen SEND; a veces se utiliza como «LINE FEED».

En los casos descritos la tecla «MP ON» debe estar desapretada.

«ROLL»: la apretadura de esta tecla permite escribir información de tal manera que el renglón nuevo resulta estar debajo del anterior. Si en la pantalla están rellenos todos los renglones entonces, al intentar escribir información nueva, todos los renglones se elevarán en un renglón, el primero de éstos desaparecerá y el último se dejará libre para la recepción.

«REPT»: al apretar esta tecla simultáneamente con cualquiera otra el código de la última se da con frecuencia de 10 Hz.

«UL»: si está apretada esta tecla todos los símbolos que se marcan aparecen en la pantalla subrayados.

La tecla «CTRL BLINK» sirve para la exposición de los símbolos que se almacenan en la memoria y que no se ven en la pantalla. Por ejemplo, LINE FEED se representa como la letra J, HOME se representa como L; estos símbolos (J y L) se reflejan en la pantalla con frecuencia de 5 Hz.

2.5. Trabajo en el terminal

En el sistema operacional Dubná en calidad de terminales se pueden utilizar el display Videotón 340, los teletipos y las máquinas de escribir Cónsul. Describiremos detalladamente el trabajo con el display Videotón 340.

El programador realiza su trabajo desde el terminal por sesiones en el *régimen de diálogo*. El ordenador reacciona a las operaciones que el usuario realiza en el terminal dando informes que aparecen en la pantalla. A su vez, el usuario compone con el teclado instrucciones que la máquina cumple.

Comienzo de la sesión.

1. El Videotón 340 se debe conectar a la red, para lo que se aprieta la tecla «POWER». Una vez que en la pantalla aparezca el trazo se debe apretar «ON LINE». El terminal está dispuesto para el trabajo.

2. Apretar la tecla «blanco». El sistema enviará a la pantalla:

MULTITIPO (FECHA DE LA VERSION/
PASS:

3. Como respuesta se debe componer la misma palabra clave (código) que usted utiliza en la tarjeta de mando *PASS, y apretar la tecla de ejecución «LF». El ordenador hará secreto su código y enviará estrellitas ****** a la pantalla *****. Si usted tarda en marcar el símbolo inmediato (más de 5 segundos) el ordenador interrumpirá la sesión y enviará a la pantalla la señal

FIN.

Si el código está erróneamente compuesto, en la pantalla aparecerá el aviso:

¡TAL NO EXISTE!
FIN.

Se debe comenzar la sesión de nuevo.

4. Si la palabra clave se ha compuesto correctamente (la lista de claves se almacena en la memoria del ordenador electrónico) entonces en la pantalla aparecerá

AMORTIGUADOR:

Como respuesta se debe componer la información respecto a aquel dispositivo amortiguador con el que usted piensa trabajar. En calidad de amortiguador puede servir o bien una cinta magnética o un fichero en el disco.

La información respecto a la cinta se compone de la misma forma que se pide en la tarjeta *TAPE: el número de la bobina, luego el símbolo / y el nombre de la cinta. Por ejemplo, cinta en la bo-

bina 135, su nombre es ABC155. Entonces en la pantalla se enciende

AMORTIGUADOR: 135/ABC155

Toda instrucción debe terminar apretando la tecla «LINE FEED».

Si en calidad de amortiguador se utiliza el disco entonces, después de AMORTIGUADOR se debe componer: un blanco obligatorio, luego el número del paquete del disco, el símbolo /, el nombre del disco, el apellido del dueño del fichero, el nombre del fichero, W.

Por ejemplo, paquete del disco 153, nombre del disco MOSCOW, apellido del dueño PETROV, nombre del fichero DUBNA1.

AMORTIGUADOR:␣153/MOSCOW, PETROV, DUBNA1, W («LINE FEED»)

Después de apretar la tecla «LINE FEED» la instrucción se cumple, pero puede aparecer la diagnosis:

NO HAY FICHERO: es posible que la información sobre el amortiguador se ha compuesto erróneamente;

EL FICHERO ESTA OCUPADO: o el disco o la cinta que usted indica están ocupados (trabaja alguien con ellos);

NO HAY CM: o bien su cinta magnética no ha sido instalada por el operario del ordenador o bien hay un error en la composición.

Instrucciones que se componen con el teclado.

1. *Instrucción ENTRADA␣N, M*, donde N y M son números octonarios. De acuerdo con esta instrucción la información que se compone luego (desde el siguiente renglón de la pantalla) ingresa en la zona número N de su amortiguador y puede ocupar un sector hasta la zona M inclusive.

Por ejemplo, ENTRADA 1,4: la información puede ocupar desde la 1ª hasta la 4ª zona del amortiguador. Si no se indica el número M, es decir, la instrucción tiene el aspecto ENTRADA N, entonces la información puede ocupar todo el amortiguador, comenzando desde la zona N.

Si usted comenzó la sesión y la máquina envió de por sí a la pantalla ENTRADA, marque usted N, M y apriete «LINE FEED». El trazo de la pantalla pasará a un renglón nuevo, en el que aparecerá un número octonario de tres cifras 001, que es el número del renglón de información.

En caso de errores al componer esta instrucción puede aparecer la diagnosis:

ER. DEL FIN

ER. DE COMPOSICION

ER. DE AMORTIGUADOR

Hay que comprobar la justeza de los números señalados de las zonas (N y M) y componer de nuevo ENTRADA␣N, M.

2. *La instrucción ENTRADA + N, M* prosigue el registro de información en el sector que comienza desde la zona N (en la que antes ya se había inscrito una parte de la información). Funciona como la instrucción ENTRADA N, M, con la diferencia de que la numeración de los renglones comienza no desde 001, sino desde el número del último renglón antes registrado.

Al componer el texto para inscribirlo en el amortiguador termine cada renglón incompleto apretando la tecla «LINE FEED». Como respuesta el trazo pasará a un nuevo renglón y se colocará después del número octonario del renglón inmediato, dado por la máquina.

Recuerde que con esto las tarjetas de mando se deben marcar inmediatamente después de que en la pantalla aparezcan los números del renglón y del blanco, es decir, desde la 5ª posición del renglón, y los operadores del Fortran desde la posición 11 (5+6). Por ejemplo:

```
055␣␣␣␣␣␣␣␣END
056␣* EXECUTE
057␣* END FILE
```

Si usted compone en el teclado las tarjetas de mando el ordenador le proporcionará un «apunte»: el comienzo de la siguiente tarjeta de mando. Por

ejemplo, en la pantalla después de * NAME aparece * PASS, etc.

No olvide apretar «LINE FEED» después de componer cada tarjeta. Si ha descubierto un error en el texto compuesto usted puede actuar de la manera siguiente. Si el renglón en el que se ha cometido el error no está todavía terminado con la apretadura de «LINE FEED», apriete entonces la tecla «DL». Con ello el ordenador «olvidará» el renglón compuesto (pero lo dejará en la pantalla del display). Si usted aprieta la tecla «→» el ordenador «recordará» un símbolo y lo visualizará en la pantalla. La apretadura reiterada de la tecla «→» restituye el renglón «olvidado» símbolo tras símbolo. Así se pueden restituir todos los símbolos correctos y llegando al error, corregirlo mediante la apretadura de la tecla correspondiente.

Si usted reveló un error en el renglón del texto después de haber apretado «LINE FEED» éste se puede corregir con el procedimiento que a continuación se describe.

3. Para concluir la introducción de información es menester, en el nuevo renglón de la pantalla sin marcar un solo símbolo, apretar «LF» (introducción de un renglón en blanco).

4. *La instrucción LIST␣N* permite leer la información comenzando desde la zona N del amortiguador. Esta información aparece en la pantalla íntegramente, ocupe lo que ocupe.

Como respuesta a la instrucción LIST N la máquina comienza a enviar a la pantalla la información de la zona N. Si en el teclado está apretada la tecla «ROLL» y la tecla «MP ON» no está oprimida entonces, al llenar la pantalla (16 renglones), toda la información se desplaza hacia arriba en un renglón, y el renglón inferior se libera para la recepción del renglón siguiente de la información. Si la posición de las teclas «ROLL» y «MP ON» es otra entonces los renglones de la información que ingresa llenan la pantalla siempre de arriba hacia abajo, es decir, después del 16º renglón de la pantalla se llena el primer renglón.

Si usted desea cesar la extracción ulterior de información a la pantalla puede apretar la tecla

«LF» y la extracción se interrumpirá. Se puede actuar análogamente si no le dio tiempo a leer o analizar la porción de información que se despachó a la pantalla. Apretando la misma tecla se puede continuar la extracción.

5. Si usted quiere ver los renglones con numeración puede hacer uso de la instrucción LIST+N. Esta instrucción se cumple de manera análoga a la instrucción LIST␣N, pero los renglones se dan con aquellos números que les fueron atribuidos durante la introducción.

6. *La instrucción KSA* permite obtener información respecto a los problemas que se encuentran en el fichero de salida.

El ordenador, como respuesta, da lo siguiente:

⟨estado⟩ ⟨número de SA⟩ ⟨número de EN⟩ ⟨nombre⟩

Aquí ⟨estado⟩ es un símbolo que indica el estado del problema:

*, el problema está resuelto y los resultados ya se han dado,

/, el problema ya pasó al fichero de salida,

␣, el problema lleva al DIAN;

⟨número de SA⟩ y ⟨número de EN⟩ son los números ordinales en los turnos de salida y entrada,

*⟨nombre⟩, contenido de la tarjeta *NAME.

7. *La instrucción PE3* ⟨número de SA⟩ permite revisar el despacho de su problema.

La instrucción PE3 + ⟨número de SA⟩ completa el despacho de la información con la numeración de los renglones.

8. Si usted quiere acelerar el ojeo de la información en el display puede dejar pasar una parte del material empleando la instrucción

HASTA ␣ ⟨imagen⟩

Aquí ⟨imagen⟩ está compuesta por no más de seis símbolos, que corresponden a los primeros símbolos de aquel renglón por el que comienza la información en la pantalla. Los blancos se ignoran. Por ejemplo,

en la zona N° 5 del amortiguador hay un paquete del programa:

```
*NAME  PETROV
*PASS: P23456
*TIME: 00.05
       PROGRAM T
       ...
       PR = H**2
       ...
       END
*EXECUTE
*END␣FILE
```

Si antes de la instrucción LIST 5 se marca HASTA*, entonces el despacho comenzará por la tarjeta *NAME,

si se marca HASTA*T entonces comenzará por la tarjeta *TIME

si se marca HASTA PR entonces comenzará por la tarjeta PROGRAM T,

si se marca HASTA PR — , entonces comenzará por la tarjeta PR=H**2,

si se marca HASTA*EN entonces comenzará la por tarjeta *END␣FILE,
etc.

La instrucción HASTA se puede componer antes de la instrucción LIST. Entonces, primero se busca el renglón cuyos primeros símbolos coincidan con ⟨imagen⟩, y luego se extrae información a la pantalla desde este renglón.

Si usted quiere puede interrumpir varias veces la extracción (apretando la tecla «LF») y emplear las instrucciones HASTA con las ⟨imágenes⟩ correspondientes.

La instrucción HASTA sin ⟨imagen⟩ realiza la búsqueda por la ⟨imagen⟩ anterior. Si no existe un renglón los primeros símbolos del cual coinciden

con la ⟨imagen⟩ entonces se da la diagnosis NO HAY IMAGEN.

La salida a la pantalla se puede interrumpir apretando la tecla «LF», y continuar apretando de nuevo «LF».

9. *La instrucción HASTA H ␣ N* permite dejar pasar los primeros N—1 renglones de la información y comenzar la proyección desde el renglón N.

Nota. Las instrucciones *HASTA* y *HASTA H* no pueden volver el texto retrocediendo.

10. Si en la zona número N hay un paquete del problema entonces éste se puede poner en marcha mediante la *instrucción ARRANQUE ␣ N.* Como respuesta aparecerá la comunicación «CALCULO», si es que éste comenzó.

11. Con la *instrucción TNP ␣ N* el problema se puede poner en marcha y comenzar el cálculo. Al poner en marcha el problema con la instrucción TNP ␣ *N* el resultado se despacha a la pantalla del terminal, y no al DIAN.

Nota. Si la máquina ya ha comenzado el cálculo del problema no se puede utilizar entonces el amortiguador, ya que éste se encuentra a disposición del problema. Una vez terminado el cálculo se puede continuar la sesión. Al ser así la máquina pedirá marcar de nuevo el amortiguador.

12. Con la *instrucción QUE TAL* se puede saber lo que ocurre con el problema en la máquina. En respuesta el ordenador comunica:

a) SH ⟨número⟩: ⟨t com.⟩ — ⟨t cál.⟩; ⟨comunicado⟩ ⟨nombre⟩. Aquí ⟨número⟩ es el número del código (del canal) en el que se calcula el problema, ⟨t com.⟩ — ⟨t cál.⟩ son el tiempo comercial y el tiempo de cálculo,

⟨comunicado⟩ es el aviso del sistema respecto a las causas de la interrupción del cálculo (por ejemplo, NO PREPARADO MF, etc.),

⟨nombre⟩, es la información de la tarjeta *NAME.

Si el usuario ha puesto en marcha varios problemas, la información que se da es respecto a todos ellos.

b) NO HAY PROBLEMA, si el cálculo se ha terminado o el problema no cayó en el turno.

Los resultados del cálculo se despachan al DIAN igual que si el problema hubiese sido introducido con tarjetas perforadas.

13. *La instrucción DIAN␣M, K* permite inscribir en el amortiguador todo aquello que el problema comunica al DIAN. Aquí M...K son los números octonarios de las zonas del amortiguador, a donde la máquina enviará el resultado de su problema. En este caso usted no obtendrá el listing (protocolo), pero con LIST M podrá ver su contenido en la pantalla. La instrucción DIAN M, K rige hasta la instrucción siguiente DIAN N, L, o hasta el final de la sesión. Con la instrucción DIAN, sin indicar N y L, se puede pasar a trabajar dando los resultados en el listing.

Ejemplo. El paquete del problema se ha compuesto en la zona número 5, y los resultados se deben alojar en la 7ª zona. Escribamos las instrucciones siguientes:

```
DIAN 7
ARRANQUE 5
QUE TAL
```

Supongamos que la máquina contestó

```
NO HAY PROBLEMA
```

Para ver el resultado hay que componer

```
LIST 7
```

Los renglones largos del DIAN (más de 80 símbolos) se extraen en dos renglones del display. La segunda parte del renglón del DIAN se marca en la pantalla con el índice de continuación ⇒.

Para reducir la extracción es razonable hacer uso de la instrucción HASTA ⟨imagen⟩. Por ejemplo,

```
HASTA*EXECU
LIST 7
```

De lo contrario en la pantalla aparecerá el contenido de todo el listing, incluyendo el apellido de usted (con letras mayúsculas).

14. Con ayuda de la *instrucción FOR ⟨lista de las especificaciones⟩* se puede acelerar la revisión de la información con texto. Esta instrucción controla el formato de los renglones de texto que se despachan. En la lista las especificaciones se enumeran mediante la coma. Existen tres tipos de especificaciones.

S, en la que cualquier grupo de blancos se sustituye por un blanco en la parte del renglón situada a la izquierda del último símbolo diferente del blanco; semejante renglón «apretado» se completa por la derecha hasta 80 símbolos con blancos. Por ejemplo, el renglón de 80 símbolos

␣␣␣␣␣␣A␣␣␣BC␣␣␣␣␣D....

después de apretarlo según la especificación S tendrá el aspecto:

␣A␣BC␣D...

Xn, en la que no se extrae a la pantalla n símbolos correspondientes de cada renglón. Si no se indica n entonces no se extrae toda la parte restante del renglón.

An, en la que se despachan sin variación n símbolos correspondientes de cada renglón. Si no figura n la especificación rige hasta el final del renglón.

Ejemplo. Supongamos que en la zona 15 se encuentra el paquete del problema:

```
*NAME PETROV
*PASS:P23456
*TIME:00.05
      PROGRAM T
      ...
      END
*EXECUTE
*END FILE
```

a) examinemos los primeros 5 símbolos del paquete, apretando el grupo de blancos:

```
FOR S A5, X
LIST 15
```

En la pantalla se visualizará:

```
*NAME
*PASS
*TIME
␣PROG
  ...
␣END
*EXEC
*END
```

b) extraigamos solamente desde el 6º hasta el 11º símbolos de cada renglón, dejándolos invariables:

```
FOR X5, A6, X
LIST 15
```

El texto que se despacha será el siguiente:

```
PETROV
:P2345
:00.05
␣PROGR
  ...
  END
UTE
FILE
```

Si en la lista de las especificaciones se ha cometido un error, en la pantalla aparecerá solamente la parte correcta de la lista. El usuario puede complementarla.

Mediante la instrucción FOR se puede rehusar de la extracción en formato.

15. *Con la instrucción PN* se puede enviar al perforador del ordenador la información de texto de la zona con número N. Por ejemplo, cumpliendo la instrucción P15 obtendremos en tarjetas perforadas el paquete del problema del ejemplo anterior.

16. *La instrucción TEL (texto arbitrario)* transmite un texto arbitrario al tablero de mando del operario del ordenador.

Por ejemplo, TEL TENGA LA BONDAD DE PONER LA CINTA 135.

17. *Con la instrucción SEND␣N, M* se puede introducir en el ordenador electrónico no sólo un símbolo, sino toda una colección de éstos. Aquí N y M son números octonarios de las zonas que limitan el sector del amortiguador (cinta, disco) destinado para la información que se introduce.

La información destinada para la transmisión en el ordenador puede componerse en el teclado, o también leerse previamente en cierto sector del amortiguador con ayuda de la instrucción LIST␣K.

En caso de composición de texto en el teclado del display es necesario realizar las siguientes operaciones.

a) Dar a la máquina la instrucción SEND␣N, M.

b) Apretar la tecla «OFF LINE» (paso al régimen autónomo) y, si es necesario, «ERASE» (borrado de la pantalla, desplazamiento del trazo a la primera posición del primer renglón).

c) Componer la información.

d) En caso de que revele un error utilizar las teclas «IC» (*insert character*), «DC» (*delete character*), «IL» (*insert line*), «DL» (*delete line*), desplazando el trazo mediante la pulsación de las teclas «↑», «→», «↓», «←».

e) Después de la verificación y corrección del texto colocar el trazo después del último símbolo y apretar la tecla «ETX». Con esto, en la memoria del display se introduce el índice del fin de la información que se transmite. Este símbolo no se visualiza en la pantalla.

f) Poner el trazo debajo del primer renglón y apretar la tecla «SEND»; con esta operación entra en el ordenador toda la información desde el renglón señalado por el trazo hasta el símbolo «ETX».

Una vez terminada la transmisión, el display permanece en la instrucción SEND y se puede transmitir la continuación del texto anterior cumpliendo todas las operaciones desde el punto b).

g) Después de transmitir todo el paquete componer la instrucción %%%. Si, luego, el display no entra en el régimen de diálogo, apriete las teclas «OFF LINE», «ERASE» y tres veces la tecla «%».

Si usted quiere transmitir información (o una parte de ella) desde la zona número K a la zona número N, realice entonces las operaciones siguientes.

1) Mediante la instrucción LIST␣K lea en la pantalla la información requerida de la zona K (apretando la tecla «blanco» usted puede interrumpir la extracción). Tenga en cuenta que un renglón de la pantalla debe quedar libro.

2) Dé la instrucción SEND␣N, M (desde la primera posición del renglón).

3) Apriete la tecla «OFF LINE». Redacte la información (si ello es necesario). Al final del texto ponga el símbolo ETX.

4) Ponga el trazo de la pantalla debajo del primer renglón del texto que se transmite y apriete la tecla «SEND».

5) Componga la instrucción %%% (véase g).

6) Para transmitir la continuación del texto de la zona K es necesario cumplir todas las operaciones, comenzando por el p. 1).

18. Con la instrucción SEND+N, M se puede continuar, en las sesiones ulteriores, la inscripción de información en la colección situada en la zona N. La diagnosis con motivo de SEND tiene el mismo sentido que la diagnosis con motivo de ENTRADA. Una vez inscrita la información en el amortiguador, ésta ya no se puede corregir en el régimen OFF LINE. Los procedimientos de redacción, que se utilizan en este caso, se describen más adelante.

19. La sesión se debe terminar con la *instrucción PARE*. Si esta instrucción no es la última en la sesión entonces, durante el trabajo del usuario siguiente, se puede estropear la información en el amortiguador dado.

20. Con la *instrucción RENUNCIACION* se puede quitar desde el terminal el problema puesto en cómputo.

21. Si en el programa Fortran puesto en cómputo se utiliza el subprograma-función IFPULT

entonces la información para éste se transmite con la *instrucción PUL␣I*, donde I es la información (constante octonaria) que se compone de la misma forma que para la elaboración en paquete.

Ejemplo. Supongamos que el programador que tiene PASS PET 456 quiere utilizar en calidad de amortiguador la cinta magnética con nombre ABC123, enrollada en la bobina 135. El fin de esta sesión es el introducir en la 5ª zona de la cinta la información: 13.567, 0.125, 315.9 y poner en cómputo el paquete que se encuentra en la 11ª zona de la cinta.

Para distinguir la información que despacha la máquina de la información que compone el programador esta última la destacaremos con cursiva. La denominación de las teclas que no son alfanuméricas y que se aprietan por el programador en el proceso de trabajo las tomaremos entre paréntesis.

␣
MULTITIPO /20/02/85/
PASS: *PET456* (LINE FEED)
AMORTIGUADOR: *135/ABC123*, W (LINE FEED)
SEND␣*5*, *10* (LINE FEED) (OFF LINE) (ERASE)
13.567 (LINE FEED)
0.125 (LINE FEED)
315.9 (LINE FEED)
(ETX) (↑) (↑) (↑) (SEND)
%%% (LINE FEED)
LIST 5 (LINE FEED)
13.567
0.125
315.9
ARRANQUE 11 (LINE FEED)
CALCULO
QUE TAL (LINE FEED)

NO HAY PROBLEMA
PARE (LINE FEED)
FIN

22. Si ponemos en marcha el problema para el cómputo mediante la *instrucción TEP⊔N* (N es el número de la zona que contiene el paquete), entonces se puede organizar el diálogo entre el problema y el terminal.

Para el intercambio de información con el terminal sirven los siguientes operadores del programa Fortran:

CALL TEROUT (A, n), que es el envío de un renglón de información en el código ISO del problema al terminal;

CALL TERIN (A, N), que es la recepción de un renglón de la información del terminal.

Aquí A es el nombre de la colección en la que se encuentra el texto transmitido (recibido), n, el número de células que ocupa este texto, N, una variable cuyo valor, una vez recibido el texto, será igual al número de palabras de máquina recibidas.

Durante el acceso a TEROUT (A, n) el número n debe darse en el programa; durante el acceso a TERIN (A, N) se debe indicar la variable N.

La aparición en la pantalla del signo «I» significa la preparación del ordenador para recibir la información consecutiva para TERIN. Esta información se compone inmediatamente después del signo «I».

Un ejemplo de programa que trabaja en el régimen de diálogo con el terminal es el sistema de información para la biblioteca de programas estandartizados. Si usted quiere utilizar este sistema introduzca entonces en la zona N del amortiguador el siguiente paquete del problema:

*NAME...
*PASS...
*TIME...
*DISC...
*FILE...

```
*LIBRARY:2
*MAIN INFLIB
*EXECUTE
INST.
*END FILE
```

Aquí en las tarjetas *DISC y *FILE se indican el disco y el fichero en el que está inscrito el sistema de información.

Una vez compuesto el paquete se debe poner en cómputo mediante la instrucción TEP N. La advertencia para el trabajo ulterior se dará en la pantalla del display. La aparición del signo «l» significa que el sistema está preparado para realizar la recepción de sus instrucciones.

23. *La instrucción OTL␣N* permite realizar la puesta a punto del programa inscrito en la zona N del amortiguador. Expongamos algunas instrucciones de la puesta a punto:

CHIS A, B, que es el envío a la variable A del programa B de un número decimal, compuesto en el renglón siguiente (después de CHIS A, B) según la especificación E,

DES A, B, que es el envío a la pantalla de los valores de la variable A,

OCHT A, B, que es la parada A,

OZA A, B, que es la parada para inscribir en la variable A,

IDI A, B, transferencia del mando a la marca A,

IDI, que es la continuación de la ejecución del programa interrumpido por la puesta a punto,

OSA A, B, es la parada por la marca A.

Aquí B es el nombre del programa (subprograma) que se pone a punto, y A, el identificador o marca en este programa.

Durante las paradas OSA, OCHT y OZA se envía a la pantalla una comunicación en la que se indican la dirección de la parada, la instrucción en la que sucedió la parada.

Si A es el nombre del COMMON-bloque o del subprograma, entonces B debe ser el símbolo *. Si A es una variable, empleada en el progra-

ma P, entonces B es el nombre de este programa.

B siempre es una precisión (por ejemplo, el nombre del programa). Si B es el número N + 1, entonces éste se interpreta como un desplazamiento de A (el N-ésimo elemento de A).

Por ejemplo, el contenido del 28 elemento del COMMON-bloque C se puede leer con la instrucción

DES␣*C*, 29

Si se ha omitido B entonces se conserva el último valor de B de las instrucciones anteriores.

Notas.

1. El nombre de los COMMON-bloques se completa con estrellitas por la izquierda y por la derecha.

2. La marca del operador va acompañada por una estrellita a la izquierda (la marca 5 pasa a ser *5).

3. La marca del operador FORMAT se complementa por la izquierda con una raya oblicua (la marca 6 pasa a ser /6).

4. Los números, que se inscriben por la instrucción CHIS deben tener el aspecto ±m.n ±Ep.

5. Si se supone poner a punto el problema desde un terminal y tener con esto acceso a las variables locales (no sólo COMMON) entonces, entre las tarjetas de mando, deberá estar presente la tarjeta

*CALL␣DEBUGER

Ejemplo. Examinemos la sesión de puesta a punto del programa desde un terminal. Supongamos que en la zona 5 del amortiguador existe el paquete:

```
*NAME...
*PASS...
*TIME...
*CALL␣DEBUGER
      PROGRAM T
      A=1.5
    1 DO 2 I=1, 10
      A = A/I
```

```
      2 B = A + B
      3 CONTINUE
        END
*EXECUTE
*END␣FILE
```

En el proceso de puesta a punto se supone:

1) despachar el contenido de B una vez terminado el ciclo;

2) insertar en la célula A el número 2,15;

3) transferir el mando al operador con la marca 1;

4) despachar el contenido de B una vez terminado el ciclo.

En la pantalla todas las operaciones se reflejarán de la manera siguiente:

...

OTL 5

CALCULO

PARADA HASTA NK 01000...

*OSA *3, T*

ADR. NNNNN *3 ...

NNNNN es el comienzo del operador con la marca 3

IDI

PARADA HASTA NK NNNNN

DES␣B

±m.n. ±Ep (número decimal — contenido de B)

CHIS A

1 + 2.15E + 0

*OSA *3*

ADR. NNNNN *3

IDI␣*1

PARADA HASTA NK NNNNN

DEC␣B

±m.n. ±Ep (número decimal — contenido de B)

IDI

...

Ejemplo. Su problema fue «arrojado» antes de la máquina con la diagnosis CONTR. INST. (control de la instrucción). Esto puede ocurrir si en lugar del programa cayeron números. La dirección NNNNN, dada en el listing conjuntamente con la diagnosis, corresponde a aquella célula de la memoria en la que resultó encontrarse el número en lugar de la instrucción del programa. Es evidente que se debe determinar qué operador del programa inscribe el número en la celda NNNNN. Para ello compongamos las instrucciones:

OTL␣M

OZA NNNNN

En la pantalla se visualizará

PARADA NK XXXXX

Aquí XXXXX es la dirección de la instrucción que «echa a perder» la célula NNNNN. Por el listing del funcionamiento anterior se puede determinar a qué operador pertenece la dirección XXXXX.

24. Si el problema que se calcula en el ordenador no fue introducido desde un terminal y no está destinado para trabajar con éste entonces, con ayuda de la *instrucción TER*, se puede pasar este problema al régimen de puesta a punto desde el terminal.

La instrucción TER permite con facilidad encontrar tales errores cuya búsqueda, generalmente, hace gastar mucho tiempo tanto al usuario como al ordenador. Expongamos ejemplos de utilización de esta instrucción.

Ejemplo. Usted tiene el listing del funcionamiento anterior del problema, del que se deduce que el problema no puede salir del ciclo. En el momento dado su problema se calcula en el ordenador. Una vez cumplidas las órdenes

TER

PASO

el ordenador enseñará en la pantalla

PARADA.NK NNNNN

donde NNNNN es la dirección de la instrucción del programa que se cumple en el momento de la respuesta de la instrucción PASO. Esta instrucción es muy probable que se encuentre en el interior de un ciclo programado incorrectamente. Conociendo esta dirección (NNNNN) y comparándola con la información dada en el listing durante el accionamiento del programa (dirección del comienzo del programa y las direcciones relativas de todos los operadores), se puede determinar el operador al que le corresponde la dirección NNNNN. Repitiendo la instrucción PASO se puede localizar el error con suficiente precisión.

25. Si el problema salió del régimen de diálogo, para restituir éste, es necesario dar la *instrucción TER.*

26. Si se requiere desocupar el terminal que se encuentra en régimen de diálogo con el problema es menester dar la *instrucción RET*.

Redacción del texto inscrito en el amortiguador desde el terminal. Además de los medios generales para la redacción de los textos el sistema tiene medios especiales, que solamente se utilizan al trabajar desde el terminal.

Como ya se dijo antes todos los renglones del texto, al introducirlos en el amortiguador del terminal, se numeran por el sistema. A cada renglón se le atribuye a la izquierda un número octonario de tres cifras (001, 002, 003, etc.). Estos números se pueden ver si se lee la zona N mediante la instrucción LIST+N.

Existen las siguientes instrucciones de redacción.

1) .=M (M es el número de la zona en la que se inscribirá la información redactada).

2) .+K (significa insertar después del renglón con el número K la información compuesta con el teclado. Los números de los renglones son octonarios).

3) .— K, L (borrar del texto los renglones desde el K° hasta el L° inclusive o insertar la información

pulsada en el teclado. Si solamente se borra el renglón K-ésimo, entonces L se puede omitir (.—K).

4) .RN (N es el número de la zona comenzando desde la cual está ubicado el material a redactar). Esta instrucción debe preceder a las instrucciones tipo 2) y 3) y puede no figurar solamente en aquel caso cuando las instrucciones de redacción se encuentran al final de la información que se redacta.

5) La instrucción ... significa el fin de las órdenes de redacción.

Para poner en marcha el cálculo del problema es necesario cumplir la instrucción

ARRANQUE K

donde K es el número de la zona que contiene el fichero de las instrucciones de redacción.

El cálculo comienza por la redacción del texto. Si el texto a redactar es un paquete del problema y está situado en la misma zona K, entonces inmediatamente después de la redacción comenzará el cálculo del problema.

6) Si en 5) no se necesita la puesta en marcha del problema para el cálculo entonces, antes de la instrucción ARRANQUE K, hay que cumplir la instrucción .S.

7) Con ayuda de las instrucciones

.LN

.S

ARRANQUE N

se puede enviar al DIAN de la máquina el contenido de la N-ésima zona del amortiguador con la numeración de los renglones. Esto es útil para la premeditación de la redacción del texto.

Si después de la redacción no se necesita la puesta en marcha para el cálculo entonces los últimos renglones de la pantalla serán los siguientes:

```
...
012.=15(LINE FEED)
.S (LINE FEED)
   (LINE FEED)
```

Redactor de pantalla. Si se necesita realizar muchas correcciones en algún texto que se encuentra o bien en la cinta o bien en el disco, es cómodo hacerlo con ayuda del *redactor de pantalla.* Daremos una breve descripción de su trabajo.

1. La instrucción RED traspasa la sesión al régimen de redactor de pantalla. Con esto el sistema envía a la pantalla el texto siguiente:

FIN
REDACTOR DE PANTALLA VERSION ⟨número⟩
#

El símboo # es el indicio de disposición para la recepción de las instrucciones de la redacción.

2. Instrucción # FILE: ⟨ID⟩, ⟨descripción⟩

Aquí ⟨ID⟩ es un identificador arbitrario (no más de 6 símbolos), ⟨descripción⟩ es la letra T para la cinta magnética, D para el disco, luego la información respecto al fichero, en la misma forma que para la instrucción AMORTIGUADOR.

Ejemplo. #FILE:A, D662/OS, BOSS, FILE,W
#FILE: B, T362/MIA, W

Si el fichero se encuentra en la cinta de la EC ЭВМ con elevada densidad de inscripción entonces éste se describe así:

H ⟨NBOB⟩ /NOMBRE DE LA CM.

Ejemplo. #FILE: C, H420/DUBNA, R

Si el fichero es de trabajo (SCRATCH), entonces S(N), W (N es el número de las zonas requeridas).

Ejemplo. #FILE, D, S(100), W

De acuerdo con la instrucción FILE el sistema encuentra en la memoria externa el fichero descrito y le atribuye el nombre dado en el ⟨ID⟩.

3. Instrucción: #EDIT: ⟨ID⟩, N

De acuerdo con esta instrucción el fichero de nombre ID, descrito antes en la instrucción FILE, se copia de la memoria externa en la N-ésima zona del amortiguador interno del redactor. El valor de N se puede elegir arbitrariamente desde 1 hasta 100.

Después de cumplida la instrucción EDIT 1 4

renglones del fichero aparecerán en la pantalla: la llamada «ventana» de la redacción.

Para que en la «ventana» aparezca un sector determinado del fichero se utilizan las instrucciones siguientes.

4. La instrucción # + adelanta la «ventana» hasta el fin del fichero (pero no más de 4096 renglones).

5. La instrucción # — adelanta la «ventana» del fichero al principio del mismo (en no más de 4096 renglones).

6. La instrucción # + NN adelanta la «ventana» hacia abajo por el fichero en NN renglones.

7. La instrucción # —NN adelanta la «ventana» hacia arriba por el fichero en NN renglones.

8. La instrucción # ⟨limitador⟩ ⟨imagen⟩ ⟨limitador⟩ adelanta la «ventana» hacia abajo por el fichero hasta el renglón que contiene ⟨imagen⟩.

Ejemplo. /.LT./ adelantará la «ventana» hasta el renglón que contenga el fragmento .LT., pero la instrucción #.LT., en calidad de fragmento, será interpretada solamente como LT, mientras el punto será interpretado como limitador. En calidad de limitador se puede utilizar cualquier símbolo que no se encuentre en el interior de la ⟨imagen⟩.

9. La instrucción #D— elimina la parte del fichero desde el renglón corriente (que señala el cursor) hasta el comienzo del fichero.

10. La instrucción #D+ elimina la parte del fichero desde el renglón corriente hasta el fin.

11. La instrucción #D—NN elimina NN renglones dede el renglón corriente hacia arriba.

12. La instrucción #D+NN elimina NN renglones desde el renglón corriente hacia abajo.

13. La instrucción #D, ⟨limitador⟩ ⟨imagen⟩ ⟨limitador⟩ elimina la parte del fichero desde el renglón corriente hacia arriba, hasta la ⟨imagen⟩ inclusive.

14. La instrucción #D+ ⟨limitador⟩ ⟨imagen⟩ ⟨limitador⟩ elimina la parte del fichero desde el renglón corriente hacia abajo, hasta la ⟨imagen⟩ inclusive.

15. Instrucción # ⟨RETURN⟩. La opresión

de la tecla «RETURN» permite pasar directamente a las operaciones de redacción de la «ventana» con ayuda de las teclas del display: DC, IC, «↑», «←», «↓», «→», «DL», «IL».

Después de que el fichero sea redactado por completo se debe apretar la tecla «RETURN». Después de esto el redactor estará preparado para la recepción de las instrucciones ulteriores.

16. La instrucción # SAVE: ⟨ID1⟩, N copia el fichero de la zona N del amortiguador interno en la memoria externa (en el fichero ⟨ID1⟩, antes descrito por la instrucción correspondiente # FILE).

17. La instrucción #END concluye el trabajo con el redactor.

Diagnosis descrita durante el trabajo desde un terminal.

1. NO EXISTE TAL o CLAVE ERRONEA se ha compuesto una clave (PASS) que no existe en la lista de las claves del ordenador. Se necesita comenzar de nuevo la sesión.

2. LA CLAVE ESTA OCUPADA: se marca una clave que en el momento dado está ocupada. Se debe pasar a otra clave o esperar hasta que ésta se libre.

3. COMPOSICION ERRONEA: la instrucción está compuesta erróneamente. Por ejemplo, el número de la zona supera el máximo admisible en el amortiguador dado, o bien el número de la zona contiene las cifras 8 y 9 (no es octonario). La instrucción debe ser compuesta de nuevo.

4. ERROR DEL AMORTIGUADOR: los límites de la zona del amortiguador están descritos erróneamente. Por ejemplo, el número de la zona inicial es mayor que el número de la zona final. Como respuesta a la interpelación de la máquina AMORTIGUADOR: se debe de nuevo encargar el amortiguador.

5. REPETICION: el amortiguador se ha marcado de nuevo, comience la sesión otra vez.

6. LA CM ESTA OCUPADA: el amortiguador dado ya se emplea por alguien (posiblemente por usted, si el problema ya se ha puesto en cálculo). Hay que esperar hasta que se libere el amortiguador y encargarlo de nuevo.

7. NO ESTA LA CM: su cinta o disco no han sido puestos por los operadores del ordenador, o usted se equivocó al marcar. Se puede pedir a los operadores que pongan la cinta.

8. EL CODIGO ESTA OCUPADO: se ha marcado la instrucción START N, S, y el código S está ocupado. Se puede enviar un comunicado a los operadores con la petición de desocupar el código, si el usuario tiene derecho a esto (instrucción TEL DESOCUPE, POR FAVOR, EL CODIGO S).

9. PROHIBIDA LA INSCRIPCION: en calidad de amortiguador se ha indicado la cinta o el disco en los que se prohíbe escribir (el régimen solamente es de lectura). Llame a los operadores del ordenador.

10. INTERCAMBIO MALO: se da o bien en caso de un mal funcionamiento de la instalación o bien al intentar leer una información inscrita con otra densidad. En este último caso primero se debe cumplir la instrucción PLO 2, y después LIST N.

La diagnosis siguiente se da al redactar el texto. Aquí el término «carta» es un sinónimo del término «renglón».

11. NO HAY IMAGEN: no se ha encontrado el renglón en el que los primeros símbolos (distintos de los blancos) coincidan con la ⟨imagen⟩ en la instrucción HASTA ⟨imagen⟩. Dé la instrucción HASTA con ⟨imagen⟩ correcta.

12. REDACCION REPETIDA EN LAS CARTAS M Y N: en los renglones M y N se ha encargado una redacción en un mismo lugar del texto.

13. NUMERO ERRONEO DE LA CARTA: se redacta el renglón con el número erróneo (posiblemente situado por debajo del renglón de la instrucción para redactar).

14. EL FICHERO DE TEXTO SE HA ESTROPEADO: el fichero no está compuesto de información de caracteres (posiblemente no sólo de caracteres).

15. EL FICHERO ESTA FORMADO ERRONEAMENTE: en el fichero existe el indicio de fin de redacción (..), pero no hay instrucciones para la redacción.

16. EL FORMATO DEL INDICADOR ES

MALO: el indicador del diapasón de las zonas en la instrucción DIAN␣N, M no sólo está compuesto de cifras octonarias (hay 8 ó 9). Componer la instrucción de nuevo.

17. INTERSECCION DE LOS DIAPASONES: en los encargos para eliminar renglones (.—N, M) hay intersección de los números. Por ejemplo,

.—003, 010

.—007, 011

18. LA TARJETA NO EXISTE: no se ha encontrado el número de la tarjeta (renglón) señalado en la tarea para la redacción.

19. TIEMPO: está componiendo la clave demasiado tiempo, o está sin trabajar en el terminal más de 2 minutos. Si el carácter de su trabajo exige premeditaciones prolongadas (más de 2 minutos entre cualesquiera dos operaciones en el terminal) debe hacer uso de la instrucción ESPERO. En este régimen el sistema no le «botará» por muy larga que sea la inactividad.

20. ERROR DEL FIN: los límites del fichero del amortiguador están indicados incorrectamente.

21. AMORTIGUADOR: después de puesto en marcha el problema para el cálculo el sistema se «olvidó» del amortiguador. Como respuesta a esta diagnosis encargue de nuevo el amortiguador.

22. NO SE APRESURE: se ha compuesto la instrucción siguiente de puesta a punto en el momento cuando todavía no se ha cumplido la instrucción anterior. Pasado algún tiempo componga la instrucción de nuevo.

23. PAGINA AJENA: está intentando durante la puesta a punto del programa revelar el contenido de una célula que no pertenece a su programa. Semejante operación no está permitida por el sistema operacional.

24. Los mensajes FICHERO OCUPADO, NO HAY PAQUETE, NO HAY ACCESO AL FICHERO corresponden a los puntos 6, 7, 9, pero se envían respecto al amortiguador en el disco.

25. NO HAY FICHERO: al encargar el amortiguador se ha compuesto un nombre erróneo.

CAPITULO III

Informaciones útiles y consejos prácticos*)

3.1. Dialectos del lenguaje Fortran

El Fortran se considera ser un lenguaje de máquina independiente. Esto debe significar que el programa escrito en Fortran sirve para cualquier ordenador que tenga traductor de este lenguaje.

Sin embargo, en la práctica, las cosas resultan ser otras. Las reglas del lenguaje Fortran para los diferentes traductores difieren un tanto entre sí. Así, pues, el lenguaje Fortran existe en forma de diversas variedades (*versiones*).

Una situación análoga se tiene también para otros lenguajes de programación. En este sentido los lenguajes de programación se comportan igual que los lenguajes conversacionales ordinarios que tienen dialectos característicos para cada región.

La aparición de diferentes versiones del Fortran está determinada por varias causas. En primer lugar, el lenguaje Fortran se desarrolla continuamente. Con esto tiene lugar su enriquecimiento con elementos

---

*) Este capítulo, a diferencia de los anteriores, está dirigido a un lector bastante preparado.

nuevos y, simultáneamente, la desaparición de aquellos elementos que no se justificaron. En segundo lugar, al elaborar un programa de traducción del lenguaje Fortran se debe tener en cuenta la específica del tipo concreto del ordenador.

En los denominados ordenadores pequeños con una capacidad de memoria relativamente pequeña y una composición reducida de operaciones de máquina, nos vemos obligados a limitarnos solamente con una parte de las posibilidades que proporciona el lenguaje Fortran.

Y, por último, en muchos casos los elaboradores de los traductores modificaron unas u otras reglas del lenguaje teniendo en cuenta las comodidades de su realización en cada traductor concreto.

La existencia de un gran número de versiones del lenguaje dificulta el intercambio de programas entre los ordenadores de diversos tipos. Por esto, para alcanzar una mayor compatibilidad de los programas en los ordenadores de distintos tipos, se emprendieron tentativas de estandartizar el lenguaje Fortran.

En 1966 en los EE.UU. se aprobó el estándar del lenguaje Fortran, que legalizó dos versiones del mismo: una para los ordenadores grandes y otra para los pequeños. La versión para los ordenadores pequeños obtuvo la denominación de *Fortran básico* (Basic Fortran), a diferencia de la versión para los ordenadores electrónicos grandes, denominada *simplemente Fortran*.

En 1978 se adoptó un estándar nuevo del lenguaje Fortran, conocido bajo el nombre de Fortran 77. Una descripción breve de las nuevas posibilidades del lenguaje Fortran, previstas por este estándar, se dará algo después.

3.2. Versiones del Fortran

Examinemos dos particularidades fundamentales de las dos versiones del lenguaje Fortran: del Fortran-Dubná y del Fortran IV. Nosotros suponemos que el lector conoce cada una de estas versiones del Fortran. Ahora existen, además de la fundamental, dos variantes optimizadoras del traductor, y asimismo los traductores del Fortran—RDA y del FOREX. Estas variantes casi no se diferencian de la fundamental por el lenguaje, pero permiten obtener una calidad superior del programa traducido.

El Fortran IV también tiene varias variantes del traductor que se distinguen entre sí por el nivel de optimización y por ciertas posibilidades de servicio a los usuarios.

El Fortran IV posee mayores posibilidades en comparación con el Fortran Dubná en el que no existen operadores de entrada-salida de acceso directo, el operador NAMELIST, asimismo los operadores RETURN i, IMPLICIT y los operadores de descripción tipo REAL*8 y análogos a él. Con esto, los operadores de entrada-salida de acceso directo y el operador NAMELIST

no tienen en el Fortran Dubná operadores equivalentes. Los operadores restantes existentes en el Fortran IV, pero que no figuran en el Fortran Dubná, permiten una sustitución equivalente mediante algunos otros operadores.

Sin embargo, en el Fortran Dubná existen los operadores ENCODE y DECODE ausentes en el Fortran IV.

Una particularidad del Fortran IV, en comparación con el Fortran Dubná, es la estandartización grande en la escritura del programa. Por esto, al pasar de la versión «más estrecha» del Fortran Dubná a la versión «más amplia» del Fortran IV surgen bastantes contrariedades determinadas por las reglas más estrictas del lenguaje Fortran IV en comparación con el Fortran Dubná.

En el Fortran Dubná se permite una mayor libertad en la utilización de las marcas en los operadores. Por ejemplo, se pueden señalar marcas en los operadores declarativos. Una misma marca puede estar indicada en un operador a ejecutar y en el operador FORMAT, por ejemplo,

5␣␣A = 3. y 5␣␣FORMAT (F10.3)

En el Fortran IV no existen estas posibilidades, y su utilización conduce a un error durante la traducción.

El Fortran Dubná prevé el título PROGRAM para el programa director. En el Fortran IV semejante título se prohíbe y el programa director no tiene título.

3.2.1. *Las constantes y su utilización*

En el Fortran IV, en comparación con el Fortran Dubná, existen ciertas limitaciones respecto a las constantes y su utilización en los operadores.

En particular, no se permiten constantes octonarias. En su lugar se pueden utilizar constantes sexadecimales. No obstante, estas últimas solamente se permiten en los operadores DATA y en los operadores de descripción del tipo, mientras que en el Fortran Dubná las constantes octonarias pueden utilizarse en los operadores al igual que las constantse de otros tipos.

El diapasón de variación de las constantes enteras en el Fortran IV es desde 0 hasta $2^{31} - 1$ (por el módulo), mientras que en el Fortran Dubná, desde 0 hasta $2^{40} - 1$. Las constantes con doble exactitud también tienen un diapasón más estrecho (desde $5 \cdot 10^{-79}$ hasta $7 \cdot 10^{+75}$). En el Fortran Dubná estas constantes tienen un diapasón desde 10^{-1232} hasta 10^{1232}.

El Fortran IV admite utilizar las constantes de texto solamente en los operadores de descripción dol tipo y en los operadores DATA, y asimismo en calidad de parámetros reales durante el acceso a los subprogramas-funciones y a los subprogramas. En el Fortran Dubná las constantes de texto se utilizan también en otros operadores al igual que las constantes do otros tipos (por ejemplo, en los operadores de atribución).

3.2.2. *Escritura de los identificadores*

Las reglas del Fortran IV no permiten la utilización de letras rusas en los identificadores. Tampoco se permiten los identificadores que contengan más de seis símbolos.

En el Fortran Dubná las letras rusas se utilizan en los identificadores al igual que las letras latinas. A los identificadores que contienen más de 6 símbolos el traductor los «reduce» hasta los primeros 6 símbolos y da una diagnosis preventiva, que no impide la salida al cómputo.

3.2.3. *Utilización de las variables con índices*

El Fortran IV prevé reglas más rigurosas de escritura de las variables con índices que el Fortran Dubná. En la variable, que es un elemento de la colección, se deben indicar tantos índices cuantos hayan en la descripción de la colección correspondiente. Solamente se admite una excepción para el operador EQUIVALENCE, donde es posible la utilización de elementos de colecciones multidimensionales como variables con un índice.

En el Fortran Dubná se pueden utilizar los elementos de las colecciones como variables con menor número de índices que se indica en la descripción de las colecciones correspondientes. Y los primeros elementos de las colecciones se pueden indicar también

sin índices. Por ejemplo, si se indica el operador DIMENSION A (10, 5) entonces en el Fortran Dubná cualquier de los operadores A (1, 1) = 0., A (1) = 0. y A = 0 será justo. En el Fortran IV solamente será correcto el operador A (1, 1) = 0.

3.2.4. Utilización del operador DATA

El Fortran Dubná permite una libertad considerablemente mayor en la utilización del operador DATA en comparación con el Fortran IV y, además, dos formas de escritura de este operador. Por ejemplo, conjuntamente con la escritura DATA X/5/, permitida por las reglas del Fortran IV, en el Fortran Dubná también es posible la escritura DATA (X = 5.).

Enumeremos las limitaciones de utilización del operador DATA en el Fortran IV en comparación con el Fortran Dubná.

1. No se pueden enviar los datos mediante el operador DATA a los elementos del COMMON-bloque no concreto.

2. El envío de los datos a los elementos concretos de los COMMON-bloques mediante el operador DATA solamente es posible dentro del BLOCK DATA.

3. El operador DATA debe situarse en el subprograma después de los operadores declarativos que describen las variables y las colecciones a donde se efectúa el envío de los datos. Por ejemplo, los operadores

```
DIMENSION A (5)
DATA A (2)/10./
```

solamente se pueden indicar en este orden (pero no en el orden inverso).

4. El tipo de las constantes que se envían deben coincidir con el tipo de las variables a donde se envían. Por ejemplo, el operador DATA R/1/ es inadmisible. En el Fortran Dubná el operador mencionado es admisible, pero no se efectúa la transformación de la constante entera 1 en tipo real.

3.2.5. Disposición de los elementos en la lista del operador COMMON y longitud del COMMON-bloque

Las reglas del Fortran IV imponen limitaciones al orden de disposición de los elementos en la lista del operador COMMON. Es necesaria, en particular, la ejecución de la condición: a cada elemento del tipo COMPLEX o DOUBLE PRECISION le debe preceder un número par de elementos del tipo REAL, INTEGER y LOGICAL. La limitación señalada está vinculada al hecho de que las máquinas tipo EC ЭВМ tienen una memoria de estructura de bytes. El cumplimiento de esta condición se asegura, por ejemplo, cuando en la lista del operador COMMON se indican primero todos los elementos del tipo COMPLEX y DOUBLE PRECISION y, después, todos los elementos de los tipos restantes.

La longitud del COMMON-bloque concreto, descrito en varios subprogramas, debe ser la misma en todos estos subprogramas.

En el Fortran-Dubná se puede, por ejemplo describir el COMMON-bloque concreto con longitud máxima en el subprograma director, mientras que en los subprogramas restantes esta longitud puede ser menor o igual a ella.

3.2.6. Disposición de los elementos en el operador EQUIVALENCE

En el operador EQUIVALENCE, igual que en el operador COMMON, tienen lugar limitaciones para la disposición de los elementos. Por ejemplo, los operadores

```
DOUBLE PRECISION A (10)
REAL B (15)
EQUIVALENCE (A, B (2))
```

dan una equivalencia inadmisible. Aquí al elemento A (1) tipo DOUBLE PRECISION le precede un elemento tipo REAL (B (1)), aunque es necesario que el número de estos elementos sea par.

3.2.7. Operador FORMAT

En el opepador FORMAT se permite una profundidad de inclusión de paréntesis igual a 2, sin contar los externos. En el Fortran Dubná la profundidad admisible de inclusión de paréntesis es igual a 3. Por ejemplo, el operador

```
6 FORMAT (2(1X, 3 (13, 2 (F10.2/5X,
  *E8.1))))
```

se puede utilizar en el Fortran Dubná, pero no se puede en el Fortran IV.

La primera posición de cada renglón no se extrae a la impresora. Esto concierne a todos los símbolos. En el Fortran Dubná la limitación señalada solamente concierne a los símbolos de control.

En el Fotran IV no existe la especificación del formato O, destinada para la entrada y salida de magnitudes octonarias. La especificación Z, que sirve para la entrada y salida de magnitudes sexadesimales, cumple funciones análogas.

3.2.8. Ausencia de los operadores ENCODE y DECODE

En el Fortran IV no existen estos operadores.

3.3. Nociones breves respecto al lenguaje Fortran 77

En 1978 en los EE.UU. fue aprobado un nuevo estándar del lenguaje Fortran. Su elaboración se terminó en 1977, por lo que obtuvo el nombre de Fortran 77.

Al elaborar el nuevo estándar se tomó en consideración la experiencia de utilización del lenguaje Fortran en los ordenadores de diferentes tipos durante 10 años. El nuevo estándar amplió considerablemente las posibilidades del lenguaje en comparación con el estándar anterior de 1966. Con esto en él se conservaron prácticamente todas las posibilidades del estándar viejo.

En el Fortran 77 se introducen datos de tipo nuevo CHARACTER, o de texto. Aunque los datos de texto se empleaban también antes en el lenguaje Fortran, no tenían un procedimiento cómodo de presentación. En el Fortran, al trabajar con datos de texto, se utilizaban palabras de máquina. Sin embargo, en dependencia del tipo del ordenador, en la palabra de máquina se aloja una cantidad distinta de símbolos. Ello conducía a la incompatibilidad de los programas para los ordenadores de diferentes tipos.

Las constantes del tipo CHARACTER son renglones de símbolos, limitados por apóstrofos, por ejemplo, 'DUBNA', 'DON"T'. En la última constante el apóstrofo doble determina el apóstrofo ordinario, es decir, esta constante se compone de los símbolos DON'T.

Toda variable de texto (o elemento de la colección) tiene una longitud fija descrita en el correspondiente operador declarativo. Si no se indica la longitud se supone que ésta es igual a 1. Por ejemplo, los operadores

```
CHARACTER*6 VOLCA
CHARACTER UNIT (20)
```

describen la variable de texto VOLGA de longitud 6 y la colección de texto UNIT, compuesta de 20 elementos de longitud 1.

Las variables y los elementos de las colecciones se pueden utilizar en el opera-

dor de atribución. Por ejemplo,

VOLGA = 'VOLGA'
UNIT (5) = 'T16'

Durante la ejecución de la operación de atribución los renglones demasiado largos se apocopan por la derecha, y los demasiado cortos se complementan con blancos por la derecha. En los ejemplos señalados la constante 'VOLGA' se complementará con un blanco por la derecha para recibir seis símbolos, y la constante 'T16' será apocopada hasta el primer símbolo, pues la variable UNIT (5) tiene longitud 1.

Para los datos de texto se introduce una operación de empalme, que se designa con dos signos //. Por ejemplo,

VOLGA = 'RT' // UNIT(8)

Aquí la variable VOLGA de longitud 6 incluirá en sí los símbolos RT, luego la variable UNIT (8), compuesta de un símbolo, y 3 blancos más.

Se puede separar una parte del renglón, indicando los números de los símbolos inicial y final. Estos números se separan por dos puntos y se ponen entre paréntesis ordinarios. Por ejemplo.

UNIT (10) = VOLGA (5 : 5)
VOLGA (3 : 6) = 'ABCD'

En el primer ejemplo la variable UNIT (10) obtiene el valor de 5° símbolo de la variable VOLGA. En el último ejemplo en la variable VOLGA el primer

y segundo símbolos permanecerán siendo los anteriores, mientras que los 4 símbolos restantes se sustituirán por ABCD.

Los renglones de símbolos se pueden comparar entre sí mediante las operaciones de relación. Con esto se supone que los símbolos están ordenados de cierto modo (por ejemplo, según el código ISO) y que el símbolo «blanco» precede a cualquier otro símbolo.

Por ejemplo, la expresión 'FOUR'.LT. 'FOURTEEN' es cierta, mientras que la relación '␣3'.GT.'2␣' es falsa.

En los operadores COMMON se introducen limitaciones para la utilización en ellos de datos de texto. Si el COMMON-bloque contiene magnitudes de texto entonces no puede contener magnitudes de otros tipos.

Para la transformación de un símbolo aislado en magnitud entera y viceversa se introducen las funciones estándares ICHAR y CHAR. La función INDEX (A, B) determina si el renglón A es o no una parte do renglón B y, en calidad de resultado, da o bien la posición inicial de los renglones que coincidieron o bien cero. Por ejemplo,

INDEX ('DUBNA', 'NA') = 4
INDEX ('DUBNA', 'VOLGA') = 0

En el Fortran 77 se han ampliado las posibilidades de trabajar con índices. Se permiten, en particular, los valores nulo y negativo del índice. El número máximo de índices es igual a 7. Durante la descripción de las colecciones se pueden indicar los

valores de los límites inferior y superior del índice, separando éstos con dos puntos. Si no se indica el límite inferior entonces se supone que éste es igual a 1.

Ejemplos.

```
DIMENSION A (5, —1 : 12, 8, 3 : 5)
CHARACTER CH (0 : 5, 8)*4
```

En las descripciones de las colecciones en FUNCTION y SUBROUTINE las dimensiones pueden estar indicadas en forma de expresiones aritméticas que contienen variables de tipo entero. Estas variables deben ser indicadas o bien en COMMON-bloque o bien como parámetros formales. El índice del elemento de la colección puede tener el aspecto de una expresión aritmética de tipo entero.

Una importante ampliación del lenguaje Fortran es la posibilidad de utilizar en el operador DO las expresiones de cualquier tipo en calidad de valor inicial, valor final y de paso de la variable de un ciclo. Además, después de la marca se puede poner una coma. Por ejemplo,

```
DO 5 K = 6, — 1, —2
DO 18, T = R + 5, X + SQRT (1.5 +
+Y)
```

En el primer ejemplo el ciclo se ejecutará cuando K = 6, 4, 2, 0, es decir, mientras que el valor de K sea mayor o igual a —1. En el segundo ejemplo la variable del ciclo T tiene el tipo de REAL y, en calidad de sus valores inicial y final, se dan expresiones de tipo real.

El número de repeticiones del ciclo

DO m V = V1, V2, V3

es igual a

MAX (INT ((n2 — n1 + n3)/n3), 0),

donde n1, n2, n3 son los valores V1, V2, V3 reducidos al tipo de variable V.

Al terminar la ejecución del ciclo la variable del ciclo tomará aquel valor que tendría durante la sucesiva repetición del ciclo. Por ejemplo, después de la ejecución del ciclo

```
      DO 25 N = 1, 10, 2
   25 M = N
```

el valor de M será igual al último valor de N con el que tuvo lugar la ejecución del ciclo, es decir, 9. Con esto el valor de N será igual a 11.

El esquema de ejecución del ciclo DO corresponde al que se realizó en el Fortran Dubná; al principio se comprueba la posibilidad de ejecución de la repetición inmediata del ciclo y, solamente después, se realiza su ejecución si ello es posible.

Una importante innovación introducida en el lenguaje Fortran es la construcción IF...THEN...ELSE. Está corresponde a las ideas de la programación estructural que obtuvieron divulgación de los últimos años. Estudiaremos el trabajo de IF...THEN...ELSE en el ejemplo siguiente:

```
      LOGICAL FUNCTION PRIME (N)
      IMPLICIT INTEGER (A — Z)
```

```
IF (N.LE.1) THEN
PRIME=. FALSE.
ELSE IF (N.EQ.2) THEN
PRIME = .TRUE.
ELSE IF (MOD (N, 2).EQ.0) THEN
PRIME=.FALSE.
ELSE
DO 18 DIVISR = 3,  INT (SQRT
*(REAL (N))), 2
IF (MOD (N, DIVISR). EQ.0) THEN
PRIME = .FALSE.
RETURN
END IF
18 CONTINUE
PRIME = .TRUE.
END IF
END
```

La función lógica PRIME da .TRUE. en calidad de resultado si su argumento N es un número primo. En caso contrario el valor PRIME es igual a .FALSE.

Como se ve del ejemplo, el operador IF...THEN determina el comienzo del bloque. Dentro del bloque pueden existir los operadores ELSE IF o ELSE, y asimismo otros operadores IF...THEN. El final del bloque que comienza con el operador en turno IF...THEN es el operador END IF. En el ejemplo señalado existen dos bloques encajados. El bloque exterior comienza por

el operador

IF (N.LE.1)THEN

y el bloque interior comienza por el operador

IF (MOD (N, DIVISR).EQ.0) THEN

El primero de los operadores END IF, situado en la zona de acción del operador DO, sirve de fin del bloque interior. El bloque exterior finaliza con el operador END IF situado delante del operador END.

El ejemplo dado demuestra la oportunidad de la construcción IF...THEN...ELSE durante la programación de expresiones lógicas complejas.

En el Fortran 77 han sido introducidas funciones unificadas estándares que pueden adquirir valores de distintos tipos en dependencia del tipo del argumento. Por ejemplo, la función SQRT (X) adquiere el valor de tipo real o complejo, o de tipo de doble precisión, en correspondencia con el tipo del argumento. De este modo, SQRT(2.) determina la raíz cuadrada para el valor real del argumento, y SQRT (2.D0) lo determina para el argumento de doble precisión (de manera análoga a DSQRT (2.D0)). La innovación mencionada simplifica considerablemente la traducción de los programas de los ordenadores de un tipo a los de otro tipo.

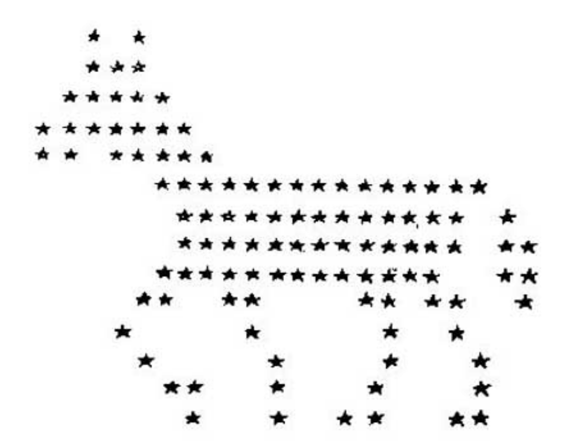
CAPITULO IV

Nuestros primeros programas

Expondremos varios programas por los que se puede comenzar el trabajo en el ordenador electrónico.

4.1. El ordenador dibuja figuras

Para hacer un dibujo con el ordenador primero se hace este dibujo con cualesquiera símbolos en papel cuadriculado, y luego se calcula en qué posiciones deben caer estos símbolos. Con ayuda de las especificaciones



de redacción nX y mH del operador FORMAT se realiza el dibujo pensado, y con el operador PRINT se transmite a la impresora.

Programa 1. Dibujo «El burrito».

```
    PROGRAM PAL
    PRINT 10
 10 FORMAT (10X, '*␣*'/10X,
   *'***'/9X, '*****'/
   C8X,'*******'/8X,'**␣*****'/13X,
   *15 ('*')/
   C14X, 13('*'), 1X, '*'/14X, 13('*'),
   *1X, '**'/
   C13X, 13('*'), 2X, '**'/12X,
  C'**␣␣**␣␣␣␣**␣**␣␣*'/11X,
   *'*', 5X, '*',
   C5X, '*␣␣*'/
   C12X,'*',5X,'*',4X,'*␣␣␣*'/13X,
   C'**␣␣␣*␣␣␣*␣␣␣␣*'/14X,
   C'*␣␣**␣␣**␣␣␣**')
    END
```

En el siguiente programa se ha utilizado el subprograma BIGTIT, que imprime con símbolos grandes cualquier texto (no más de 18 símbolos). La llamada es

CALL BIGTIT (H⟨texto⟩, M, 6H******)

Para imprimir el texto éste se divide en grupos de 6 símbolos en cada uno de ellos. Después de cada 6H se escribe el grupo correspondiente. M es la colección

de trabajo que debe describirse como DIMENSION M (144). Después de los últimos 6H se escriben los símbolos con los que escribirán las letras grandes.

4.2. Los programas imprimen las tablas de las funciones

Programa 2. Tabla de los cosenos.

```
   PROGRAM  COSIN
   DIMENSION  X(64), Y(64)
   X(1) = 0
   DO 4 N = 1,63
   Y (N) = COS (X (N))
   X (N + 1) = X (N) + 0.1
 4 CONTINUE
   PRINT 8,  ( X(1), Y (L), L = 1, 63)
 8 FORMAT (2 (4HCOS  (, F5.3, 2H)=,
  *F6.3, 3X))
   END
```

Resultados del trabajo del programa:

```
COS(0.000) = 1.000       COS(0.100) = 0.995
COS(0.200) = 0.980       COS(0.300) = 0.955
COS(0.400) = 0.921       COS(0.500) = 0.878
COS(0.600) = 0.825       COS(0.700) = 0.765
COS(0.800) = 0.697       COS(0.900) = 0.622
COS(1.000) = 0.540       COS(1.100) = 0.454
COS(1.200) = 0.362       COS(1.300) = 0.267
COS(1.400) = 0.170       COS(1.500) = 0.071
COS(1.600) = —0.029      COS(1.700) = —0.129
COS(1.800) = —0.227      COS(1.900) = —0.324
```

```
COS(2.00)  = −0.416      COS(2.100) = −0.505
COS(2.200) = −0.589      COS(2.300) = −0.666
COS(2.400) = −0.738      COS(2.500) = −0.801
COS(2.600) = −0.857      COS(2.700) = −0.904
COS(2.800) = −0.942      COS(2.900) = −0.971
COS(3.000) = −0.990      COS(3.100) = −0.999
COS(3.200) = −0.998      COS(3.300) = −0.987
```

Programa 3. Tobla de los senos, cosenos y tangentes

```
    PROGRAM TAB
    DIMENSION X (64), Y (63), Z (63),
    A (63)
    X (1) = 0.
    DO 4 N = 1, 63
    Y (N) = SIN (X (N))
    Z (N) = COS (X (N))
    A (N) = TAN (X (N))
    X (N + 1) = X (N) + 0.1
  4 CONTINUE
    PRINT 5, (X (L), Y (L), Z (L), A (L),
   *L = 1, 63)
  5 FORMAT (10X, 1HX, 6X, 5HSIN␣X,
   *8X,
   C5HCOS␣X, 8X
   C5HTAN␣X/63(10X, F3.1, 4X, F6.3,
   *7X, F6.3,
   C6X, F7.3/))
    END
```

Resultados del trabajo del programa:

X	SIN X	COS X	TAN X
0.0	0.000	1.000	0.000
0.1	0.100	0.995	0.100
0.2	0.199	0.980	0.203
0.3	0.296	0.955	0.309
0.4	0.389	0.921	0.423
0.5	0.479	0.878	0.546
0.6	0.565	0.825	0.664
0.7	0.644	0.765	0.842
0.8	0.717	0.697	1.030
0.9	0.783	0.622	1.260
1.0	0.841	0.540	1.557
1.1	0.891	0.454	1.965
1.2	0.932	0.362	2.572
1.3	0.964	0.267	3.602
1.4	0.985	0.170	5.798
1.5	0.997	0.071	14.101
1.6	1.000	—0.029	—34.233
1.7	0.992	—0.129	—7.697
1.8	0.974	—0.227	—4.286
1.9	0.946	—0.323	—2.927
2.0	0.909	—0.416	—2.185

. . .

4.3. Trabajo con las colecciones

Programa 4. Extraer de la colección M números que se dividen por 3.

```
      PROGRAM  MA
      DIMENSION M (10)
      READ  9,  M
    9 FORMAT  (10I2)
```

```
   DO 8 I = 1, 10
   IF (M(I)/3*3 — M (I)), 8, 2, 8
 2 PRINT 7, M (I)
 7 FORMAT (5X, I2)
 8 CONTINUE
   END
```

Resultado del trabajo del programa:

```
3
6
9
```

Este programa introduce diez números enteros e imprime aquellos de éstos que se dividen exactamente entre 3. La verificación de la divisibilidad se basa en lo siguiente: M (I)/3*3 es M (I) solamente para aquellas M (I) que se dividen por 3; para las restantes el resultado M (I)/3 es la parte entera del cociente M (I) : 3 y, por esto, M (I)/3*3 será menor de M (I).

Programa 5. Ordenación de una colección de 100 números según su acrecentamiento. Este programa utiliza el algoritmo que a veces se llama «de burbuja»: cada par de números vecinos se ordena de tal modo que a la derecha quede el número mayor. Como resultado el número máximo cae en la última célula de la colección, y después de la revisión siguiente de los pares el máximo de los números restantes resulta estar en el penúltimo sitio, etc. El proceso termina cuando al revisar la colección no existan dos números que haya que

cambiarlos de sitio.

```
      PROGRAM S
      DIMENSION C (100)
      READ 11, C
   11 FORMAT (10F6.3)
      CALL SORT (100, C)
      PRINT 21, C
   21 FORMAT (10 (2X, F6.3))
      END
      SUBROUTINE SORT (N, A)
      DIMENSION A (N)
      M = N - 1
   10 PR = 0
      DO 20 I = 1, M
      IF (A (I) - A (I + 1)) 20, 20, 40
   20 CONTINUE
      M = M - 1
      IF (PR) 10, 30, 10
   30 RETURN
   40 C = A (I)
      A (I) = A (I + 1)
      A (I + 1) = C
      PR = 1
      GO TO 20
      END
```

4.4. Resolución de una ecuación cuadrática

Programa 6. Resolución de una ecuación cuadrática con investigación completa.

```
   PROGRAM SQRTEQ
   READ 1, A, B, C
 1 FORMAT (3F6.3)
   IF (A) 3, 2, 3
 3 IF (B**2—4*A*C) 4, 5, 5
 5 X1 = (— B + SQRT(B**2—
  —4*A*C))/(2*A)
   X2 = (—B — SQRT (B**2—
  —4*A*C))/(2*A)
   PRINT 6, A, B, C, X1, X2
 6 FORMAT (10X, 2HA=, F6.3, 2X,
  *2HB=, F6.3,
  C2X, 2HC=, F6.3/10X, 3HX1 =,
  *F7.3, 2X,
  C3HX2=, F7.3)
   GO TO 100
 2 IF (B) 8, 7, 8
 8 X = — C/B
   PRINT 9, A, B, C, X
 9 FORMAT (4F6.3)
   GO TO 100
 7 IF (C) 4, 11, 4
 4 PRINT 10, A, B, C
10 FORMAT (10X, 2HA=, F6.3, 2X,
  *2HB=, F6.3,
  C2X, 2HC=, F6.3/
  C10X, 13HNO HAY␣RAICES)
   GO TO 100
11 PRINT 12, A, B, C
```

```
12 FORMAT (10X, 2HA=, F6.3, 2X,
  *2HB=, F6.3,
  C2X, 2HC=, F6.3/
  C 10X, 11HX—CUALQUIER)
100 CONTINUE
    END
```

El resultado del trabajo del programa es:

A = 23.333␣␣B = 15.028␣␣C = — 2.051
X1 = 0.116␣␣X2 = — 0.760

4.5. Laberinto elemental

Programa 7. «Laberinto». Este programa resuelve el problema siguiente: la colección bidimensional LAB (10, 10), compuesta de ceros y unidades, representa en sí un laberinto sin callejones sin salida y sin anillos. La pared se representa con unidades y el corredor, con ceros. La máquina debe «pasar» desde la entrada hasta la salida sustituyendo los ceros por ochos.

```
    PROGRAM MOD
    DIMENSION LAB (10, 10)
    READ 1, LAB
   1FORMAT (10I1)
    I = 1
    J = 2
```

```
 2 LAB  (I,  J) = 8
   IF (LAB (I, J + 1)) 3, 4, 3
 4 J = J + 1
   IF (J — 11)2, 6, 6
 3 IF (LAB (I + 1, J)) 5, 7, 5
 5 IF (LAB (I, J — 1)) 11, 8, 11
 7 I = I + 1
   IF (J — 11) 2, 6, 6
 8 J = J — 1
   IF (J) 6, 6, 2
11 IF (LAB (I — 1, J)) 6, 12, 6
12 I = I — 1
   IF (I) 6, 6, 2
 6 PRINT 13, ((LAB (I, J), J = 1, 10),
  *I = 1, 10)
13 FORMAT (2X, 10I1)
   END
```

El resultado del trabajo de este programa es:

```
1811111111
1811111111
1811888811
1811811811
1811881881
1811181181
1818881181
1818111181
1888111188
1111111111
```

4.6. Resolución de la ecuación $f(x) = 0$ con el método de división por la mitad

Este método permite hallar aproximadamente el cero de la función dada en el segmento [A, B] de tal manera que $f(A) \times \times f(B) \leqslant 0$. En este segmento $f(x)$ es continua y tiene un cero único.

Programa 8. Cálculo de la raíz de la ecuación $x^3 - x^2 - 2 = 0$ en el segmento [1, 2] con precisión $\varepsilon = 10^{-2}$.

```
      PROGRAM MPD
      A = 1.
      B = 2.
      EPS = 1.E-2                    Datos iniciales
      IF (F(A)*F(B)) 1, 2, 3         Investigación del
                                     signo
    1 C=(A+B)/2                      División del seg-
                                     mento por la mitad
      IF (F(A)*F(C)) 4, 7, 6
    4 B=C                            Omisión del punto
                                     de la derecha
    5 IF (ABS(A-B)-EPS)              Salida por la pre-
                                     cisión
   C7, 1, 1
    7 T=F(C)
      PRINT 9, C, T
    9 FORMAT (20X, 2HC=, F4.2, 5X, 5HF(C)=,
   *F5.3)
      GO TO 100
    2 IF (F(A)) 10, 11, 10
   11 C=A                            Si f(A)*f(B) = 0
      GO TO 7
   10 C=B
      GO TO 7
```

```
    3 PRINT 12
   12 FORMAT (20X,13HNO␣HAY␣RAICES)
      GO TO 100
    6 A=C                          Omisión del punto
                                   de la izquierda
      GO TO 5
  100 CONTINUE
      END
      FUNCTION F(X)
      F=X**3-X**2-2.
      RETURN
      END
```

El resultado del cálculo es:
C=1.70 F(C)=0.000

4.7. Algunas recomendaciones al programador debutante

Publicamos estos programas pensando que al lector debutante precisamente le serán más comprensibles aquellos programas que fueron compuestos por personas que daban los primeros pasos en la programación.

Más arriba ya escribimos que todo problema puede ser resuelto con ayuda de distintos programas, que se diferencian por el tiempo de su ejecución y por el número de operadores. Esto puede depender tanto de la calificación del programador como del destino del programa y parámetros del ordenador.

Es natural que todo programador aspira a la creación de un programa óptimo. Pero

la noción de óptimo no se determina unívocamente. Hablando en general, el programa que es más corto y más rápido es también más óptimo. Pero, frecuentemente, estos dos parámetros se encuentran en dependencia recíproca: un programa más corto trabaja más tiempo que un programa más largo. ¿A cúal de estos dos programas se debe dar preferencia? Esto depende de las condiciones concretas.

Para los ordenadores de poca memoria los programas cortos se consideran óptimos. Si en la máquina se resuelve el problema de control del vuelo de un cohete entonces es primordial la exigencia de velocidad máxima de funcionamiento del programa. Semejantes programas tienen una estructura compleja y exigen mucho tiempo para su composición y puesta a punto.

Muchos programas de control de los procesos reales no contienen ciclos: las instrucciones se escriben tantas veces cuantan se ejecutan. Con esto se reducen los «gastos accesorios» relacionados con la organización de los ciclos. En semejantes condiciones precisamente tal programa voluminoso es óptimo.

Si ante el programador se plantea el problema de componer en poco tiempo un programa que ocupe menos de una hora de tiempo de cómputo y que esté destinado para una sola pasada entonces será óptimo el programa que esté escrito rápidamente y que requiera una mínima cantidad de salidas a la máquina para su puesta a punto.

Si usted ha compuesto y ha puesto a punto semejante programa y después ha inventado un nuevo procedimiento más corto de solución ¿tiene sentido cambiar el programa «viejo?» Evidentemente no, pues la renovación y la puesta a punto harán gastar mucho tiempo de usted y de la máquina y es muy probable que, en total, no se gane tiempo.

Otra cosa es cuando el programa se destina para una utilización múltiple o exige para su ejecución mucho tiempo de máquina. En este caso es menester pensar bien el algoritmo y, en cualquier etapa de la elaboración inscribir las modificaciones que mejoren sus parámetros.

Se presentan exigencias especiales a los subprogramas estándares que calculan funciones matemáticas elementales en los ordenadores

$$\operatorname{sen} x, \ \cos x, \ \operatorname{tg} x, \ \sqrt{x}, \ \text{etc.}$$

Estas funciones se utilizan al resolver la mayoría de los problemas y funcionan millones de veces. Por ello, si estos programas no son óptimos, se pierden muchas horas de tiempo de máquina.

Ahora expondremos uno de los algoritmos más simples para el cálculo de sen x.

Cálculo de la función sen x *mediante el desarrollo en serie.* En el curso de análisis matemático se demuestra que si el ángulo se da en radianes entonces es justa la

igualdad

$$\operatorname{sen} x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots,$$

es decir,

$$\operatorname{sen} x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

es la suma de una serie infinita convergente.

Designemos mediante a_n los términos de esta serie:

$$a_0 = x, \quad a_1 = -\frac{x^3}{3!}, \quad a_2 = \frac{x^5}{5!},$$

$$a_3 = -\frac{x^7}{7!}, \text{ etc.}$$

La suma de una serie infinita convergente, con cierto error, puede ser sustituida por la suma de una serie finita.

Supongamos que queremos hallar el valor del sen x con cierta precisión ε.

Puesto que $\lim\limits_{n\to\infty} \frac{x^{2n+1}}{(2n+1)!} = 0$, se hallará un número N para el que para todas $n > N$ se cumplirá la correlación $|a_n| < \varepsilon$.

Es sabido que la suma de semejante serie infinita de términos de signo variable $\sum\limits_{n=0}^{\infty} a_n$ se puede sustituir por la suma de la serie finita $\sum\limits_{n=0}^{N} a_n$ con un error que no superará el primer término omitido

a_{N+1}, es decir,

$$\left|\sum_{n=0}^{\infty} a_n - \sum_{n=0}^{N} a_n\right| \leqslant |a_{N+1}| < \varepsilon.$$

Una vez calculado el término inmediato de la serie se comprueba la condición $|a_n| < \varepsilon$. Si no se cumple entonces a_n se añade a la suma de los términos anteriores y se calcula a_{n+1}. Si $|a_n| < \varepsilon$ entonces la suma obtenida $\sum_{n=0}^{N} a_n$ da el valor de sen x con precisión ε.

Expondremos el subprograma-función que realiza este algoritmo. De parámetros formales sirven el argumento X y la precisión EPS.

```
      FUNCTION SINUS (X, EPS)
      S = 0.
      A = X
      B = 3.
      X2 = X**2
    3 S = S + A
      A = -A*X**2/(B*(B - 1))
      B = B + 2
      IF (ABS (A) - EPS) 2, 3, 3
    2 SINUS = S
      RETURN
      END
```

Aclaremos el trabajo de este programa. En la célula S se acumula la suma $\sum_{n=0}^{N} a_n$. Su valor inicial es cero.

La célula A está destinada a la variable a_n $(a_0 = x)$; B contiene $2n + 1$.

El operador con la marca 3 añade el valor del consecutivo a_n a la suma de los términos anteriores.

El operador A = —A*X**2/(B* (B—1) calcula el término siguiente. Con esto se emplea la relación recurrente

$$a_n = a_{n-1}\left(-\frac{x^2}{2n(2n+1)}\right), \qquad n > 0,$$

es decir, el término posterior se diferencia del anterior

1) por ser de signo opuesto,
2) por ser mayor en x^2 veces,
3) por ser menor en $2n(2n+1)$ veces.

Efectivamente,

$$a_0 = x, \quad a_1 = a_0\left(-\frac{x^2}{2\cdot 3}\right) = -\frac{x^3}{3!},$$

$$a_1 = -\frac{x^3}{3!}, \quad a_2 = a_1\left(-\frac{x^2}{4\cdot 5}\right) = \frac{x^5}{5!},$$

$$a_2 = \frac{x^5}{5!}, \quad a_3 = a_2\left(-\frac{x^2}{6\cdot 7}\right) = -\frac{x^7}{7!}, \text{ etc.}$$

Para el cálculo del siguiente a_n a B se añade 2, y el operador

IF (ABS (A) — EPS) 2, 3, 3

comprueba la condición $|a_n| < \varepsilon$. En el caso $|a_n| \geqslant \varepsilon$ el salto de mando se para al operador con la marca 3 y se prosigue el cálculo de los términos de la serie.

Si $|a_n| < \varepsilon$ entonces al nombre del subprograma se le atribuye el valor de la

suma calculada y tiene lugar el retorno al programa de llamada que obtendrá el valor de sen x con precisión ε.

Ejemplo. Componer la tabla de los senos para el argumento x que varía desde 0 hasta 9,9 con paso $h = 0,1$.

La tabla la llenaremos así. La primera columna vertical la asignaremos para la parte entera del argumento x, y el primer renglón para la parte fraccionaria. Entonces los valores de la función se situarán en la intersección de la columna y del renglón, que corresponden a la parte entera y a la fraccionaria del argumento. Por ejemplo, el valor sen 3.1 lo escribiremos en el cuarto renglón y segunda columna de la tabla (pues el primer renglón y la primera columna corresponden a los valores nulos), etc.

En calidad de subprograma-función que calcula el sen x tomemos el subprograma SINUS, que antes escribimos.

```
      PROGRAM STAB
      DIMENSION TAB (11, 11)
      DO 1 K = 2, 11
      TAB (K, 1) = K − 2
    1 TAB (1, K) = (K − 2)*0.1
      TAB (1, 1) = 0
      DO 2 K = 2, 11
      DO 2 L = 2, 11
    2 TAB (K, L) = SINUS (TAB (K, 1)
     C + TAB (1, L), 1.E−6)
      PRINT 3, ((TAB (I, J), J = 1, 11),
     CI = 1, 11)
```

```
3 FORMAT (1H1, 45X
 C14H TABLA␣DE␣SENOS//
 C (2X, 11F10.6))
  END
```

El programa trabaja así. Primero se inscriben en la primera columna de la colección B (elementos B (K, 1)) los valores enteros del argumento 0; 1; 2; 3; . . .; 9, y en el primer renglón (elementos B (1, K)) los valores fraccionarios 0,0; 0,1; 0,2; 0,3; . . .; 0,9. Después de esto se atribuye cero al elemento TAB (1, 1) y se comienza a llenar la tabla con los valores del seno. En calidad de primer parámetro real sirve la expresión aritmética TAB (K, 1) + + TAB (1, L) que calcula el valor del argumento X: TAB (K, 1) contiene la parte entera y TAB (1, L), la parte fraccionaria. El segundo parámetro real contiene la precisión de cálculo.

El valor del seno del argumento cuya parte entera corresponde al renglón K y la parte fraccionaria a la columna L, se envía al elemento TAB (K, L).

Una vez llena la tabla ésta se saca a la impresora por renglones que contienen 11 elementos.

Citaremos los resultados del trabajo del programa.

0.000 000	0.000 000	0.100 000	0.200 000...
0.000 000	0.000 000	0.099 833	0.198 669
1.000 000	0.841 471	0.891 207	0.932 039
2.000 000	0.909 297	0.863 209	0.808 496

3.000 000	0.141 120	0.041 581	—0.058 374
4.000 000	—0.756 802	—0.818 277	—0.871 576
5.000 000	—0.958 924	—0.925 815	—0.883 455
6.000 000	—0.279 416	—0.182 163	—0.083 089
7.000 000	0.656 987	0.728 969	0.793 668
8.000 000	0.989 358	0.969 890	0.940 731
9.000 000	0.412 118	0.319 098	0.222 89 ...

Nos detendremos más detalladamente en la primera especificación del operador FORMAT — 1H1. Esta prescribe extraer a la unidad haste la impresora en calidad de primer símbolo del renglón. Semejante indicación se interpreta por la impresora como la instrucción de imprimir la información ulterior en una nueva página del listing.

Si en el algoritmo expuesto para el cálculo de sen x no se emplea la relación recurrente y se calcula por la fórmula

$$a_n = (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

entonces, inmediatamente, se empeoran los parámetros del programa.

En primer lugar, el sen x se calculará mucho más tiempo, pues será necesario organizar un ciclo para hallar $(2n+1)!$

En segundo lugar, para las x suficientemente grandes las magnitudes x^{2n+1} ó $(2n+1)!$ (si se calculan por separado el numerador y el denominador) pueden repletar la red de dígitos de la máquina. Pero incluso los cálculos por la fórmula recurrente no salvan el programa de la imperfección sustancial. La cosa reside en que en

el ordenador el algoritmo trabaja no para cualesquiera valores del argumento. Para x suficientemente grandes los términos de la serie al principio crecen por su magnitud absoluta y después ya comienzan a disminuir. Por ejemplo, cuando $|x| > \sqrt{6}$ el segundo término del desarrollo a_1 es mayor que el primer término (por el módulo).

Examinemos la conducta de los términos de la serie cuando x es del orden 10^3. Entonces, aproximadamente hasta $n = 500$, las magnitudes absolutas a_n aumentan con el crecimiento de n. Pero incluso si x es tal que no tuvo lugar la parada por avería no se puede estar seguro de que el resultado dado es correcto.

Por ejemplo, supongamos que $x = 10$. No es difícil calcular que $|a_5|$ del desarrollo en serie será del orden 10^4. Y puesto que como resultado de la suma de los términos de la serie los ordenes superiores (exactos) se pierden, el valor de sen x se da solamente con tres signos exactos.

Para las x grandes el número de signos exactos es aún menor. Puede resultar que el error del cálculo sea de un mismo orden que el del resultado, o incluso supere a éste.

Citemos un ejemplo concreto de cálculo en el ordenador del seno con un programa análogo. Se calculó el sen$\left(\frac{\pi}{6} + 2\pi n\right)$ para $n = 0, 1, 2, 3, \ldots, 8$ en una máquina que permitía trabajar con ocho signos exactos. El resultado se expone en la tabla 1.

30°	0.523 598 78	0.499 999 99
390°	6.806 784 15	0.499 999 93
750°	13.089 969 52	0.500 135 07
1110°	19.373 154 88	0.516 584 90
1470°	25.656 340 12	24.254 018
1830°	31.939 525 60	14 380.237
2190°	38.222 711 09	25 902 480.
2550°	44.505 896 09	−130 402 508.
2910°	50.789 081 57	−83 272 283.

En la primera columna se exponen los ángulos en grados, en la segunda en radianes, y en la tercera columna se exponen los valores de los senos, obtenidos con precisión $\varepsilon = 10^{-8}$.

Para cada argumento el valor exacto del seno es de 0,5.

De la tabla citada se deduce que ya para $x = 1470°$ el error del cálculo supera muchas veces la magnitud buscada.

Esto ocurrió porque para el ángulo de 1470° ($x = 25.656\ 340\ 12$ radianes) el término máximo de desarrollo del seno en serie es aproximadamente igual a 5 503 768 000, mientras que el cálculo se efectuó con ocho signos exactos. Por consiguiente, este término se obtuvo con un error absoluto (error de redondeo) de 50, y el valor del seno yace en el intervalo

$$24.254\ 018 \pm 50.$$

Los cálculos con precisión doble mejoran un poco el cuadro, pero no juegan un papel de principio, pues para los argumentos suficientemente grandes el error es inadmisiblemente grande. Al analizar tan detalladamente la realización programática y el cálculo en el ordenador del seno por la fórmula

$$\operatorname{sen} x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n(2n+1)},$$

queríamos demostrar al lector que la cuestión respecto a la optimación del programa está inseparablemente ligada a la cuestión respecto a la precisión del cálculo.

Todo programa debe garantizar la precisión necesaria para los distintos valores de las variables.

Al poner a punto el programa no sólo se debe comprobar si se ha realizado correctamente el algoritmo, sino también apreciar los errores que se obtienen en el ordenador con los valores extremos de las variables.

El programador debe entender bien en qué límites pueden cambiar las variables, y tener esto en cuenta al elegir el algoritmo.

Por ejemplo, la fórmula de integración de Simpson

$$I = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + \dots$$

$$\dots + 4y_{n-1} + y_n)$$

debe ser presentada como

$$I = \frac{hy_0}{3} + \frac{4}{3} hy_1 + \frac{2}{3} hy_2 + \frac{4}{3} y_3 + \dots$$
$$\dots + \frac{hy_n}{3}$$

para valores suficientemente grandes de la función, ya que de lo contrario puede tener lugar la repleción (parada por avería).

Y, al revés, cuando los valores de la función son pequeños, es mejor primero sumar y_i y luego a multiplicar por la magnitud pequeña h, para evitar la obtención del cero de máquina.

Desgraciadamente, no se pueden dar recomendaciones para todos los casos. Pero siempre hay que recordar las paradas por avería, los ceros de máquina y el hecho de que la máquina puede dar solamente un número limitado de signos exactos.

imposible una explotación normal de los ordenadores.

Entre los programas que componen el software del ordenador moderno un papel especial pertenece a aquellos que funcionan conjuntamente con el equipamiento de la máquina. Esta parte del software generalmente se denomina sistema operacional (SO). El sistema operacional es el «anexo programático» al ordenador, y se suministra por la fábrica conjuntamente con el completo de la máquina. Señalemos que bastante frecuentemente en la composición del SO incluyen también ciertos programas de servicio, y, además, programas traductores de los lenguajes de programación.

Los sistemas operacionales modernos son complejos de programas, que contienen hasta varios millones de instrucciones. Los programas que entran en la composición del SO deben ser particularmente seguros en la explotación y trabajar rápidamente. Por esto, en la composición del SO toman parte los programadores más calificados, que habitualmente se llaman programadores de sistemas.

Todos los programas restantes (que no son del sistema) se denominan aplicados. Para la realización de semejantes programas existen, precisamente, los ordenadores. Las cuestiones relacionadas con la preparación de los programas aplicados son el objeto de la *programación aplicativa.*

Entre los programas aplicados los programas de cálculo componen una gran parte

Conclusión

En este pequeño libro solamente tocamos algunas pocas cuestiones relacionadas con una amplia esfera de la actividad humana: la programación en los ordenadores de cálculo.

Ahora, cuando llegó la hora de despedirnos del lector, probemos una vez más a echar una mirada al tema del que se trataba en las páginas de este libro. Quisiéramos, en particular, examinar esquemáticamente algunas cuestiones de la programación que no tocamos en el libro.

En la programación se pueden distinguir varias partes independientes. Semejantes partes, según nuestra opinión, son la programación de sistema, la programación aplicada y la programación teórica. Reconociendo la condicionalidad de semejante división, contaremos brevemente sobre cada una de estas partes.

La programación de sistema abarca el número de problemas relacionados con la composición de programas que entran en la estructura del software. Estos son programas sin los cuales, hablando en general, es

(llamados también programas científicos o de ingeniería). La mayor parte de este libro está dedicada a los problemas relacionados con la escritura de semejantes programas.

Los programas aplicados frecuentemente se reúnen en bibliotecas que se forman, por ejemplo, por los indicios temáticos. Semejantes bibliotecas existen, prácticamente, en cada ordenador moderno.

Aunque la programación es precisamente una rama de la actividad práctica, para su desarrollo exitoso son necesarias también las investigaciones teóricas. A éstas pertenecen, por ejemplo, la elaboración de la teoría de los lenguajes de programación, de los métodos de traducción y optimación de los programas. Estos y una serie de otros problemas teóricos de programación componen el tema de la *programación teórica.*

Señalemos que las partes examinadas de la programación se separaron en el proceso de su desarrollo, que es lejos de ser terminado.

Esperamos que, aunque parcialmente, hayamos logrado dos objetivos: dar a conocer a los lectores ciertos aspectos de la programación y ayudarlos en la asimilación práctica de este tema.

24. 1. a) X = (B**2+C**2)/D
b) E = (35.*X+Y)/A**3
c) A = 3.*C/(25.*D)

2. a) $\frac{A}{B}(C+D)$, b) $\frac{A(C+D)}{B}$,

c) $\frac{A}{B(C+D)}$, d) $\frac{AC}{B}+D$

25. a) 0; b) 2; c) 1; d) 0; e) 7.

26. GO TO 100

27. F3.1, F6.3, F6.2, I2, I4, F5.1

28. PRINT 2, A, IB, C

```
2FORMAT (F5.2, I6, F7.5)
```

29. 1)

```
      PROGRAM SQ
      A= 15.7328
      H= 0.031289
      S= 0.5*A*H
      PRINT 1, S
    1 FORMAT (10X, F10.7)
      END
```

2)

```
      PROGRAM GEOM
      B= 1.135619
      Q= 0.999999
      S= B*(1. —Q**10)/(1. —Q)
      PRINT 1, S
```

```
  1 FORMAT (10X, F12.6)
    END
```

30. 1)

```
      PROGRAM ROOT
      READ 10, A, B, C
   10 FORMAT (3F7.2)
      X1 = (-B+SQRT(B**2-4*A*C))/
     *(2*A)
      X2 = (-B-SQRT(B**2-4*A*C))/
     *(2*A)
      PRINT 20, X1, X2
   20 FORMAT (10X, 6HRAICES=, E12.5,
     *3H␣I␣, E12.5)
      END
```

en el papel desde la posición 11ª se imprimirá

RAICES = (número X1) Y (número X2)

2)

```
      PROGRAM SP
      READ 15, A, B, ALFA
   15 FORMAT (3F8.4)
      S=A*B*0.5*SIN(ALFA)
      C=SQRT(A**2+B**2-2*A*B*
     *COS(ALFA))
      P= A+B+C
      PRINT 21, S, P
   21 FORMAT (20X, 7HAREA␣S=,
     *E12.5, *12HPERIMETRO␣P=,
     *E12.5)
      END
```

en el papel desde la 21ª posición se imprimirá:

AREA (fecha) PERIMETRO (fecha)

3) a)

```
      Y=SQRT(A*B)*SIN(A)*ALOG(B)
     *+ABS(SIN(A)*SQRT(ABS
     *(SIN(A)**2-COS(B)**2))
```

c)

```
      Y=A/(2*SQRT(B*SQRT(A)))
```

31.

```
      IF (A), 1, 2, 2
    1 Y=-A
```

```
      GO TO 3
    2 Y=A
    3 CONTINUE
```

32. 1)

```
      PROGRAM AK
      A=1.
      DO 10 I=2, 15
   10 A=SQRT(A)+1
      PRINT 30, A
   30 FORMAT (10X, 2HA=, E(12.5)
      END
```

2)

```
      PROGRAM SUM
      A=2
      S=2
      DO 5 I=2, 20
      A=A**2/(A+3)
    5 S=S+A
      PRINT 10, S
   10 FORMAT (10X, 2HS=, E12.5)
      END
```

33. 1.

```
      DIMENSION C(8)
      DATA C/15.2, 5*17.3, 1.5, 3.2/
```

2. a) es inadmisible la mezcla de tipos;
   b) complejo;
   c) es inadmisible la mezcla de tipos,

34.

```
      A=-B
      IF (X.GT.3)A=-B+B*SQRT(X-3)
```

Bibliografía

1. J.K. Hughes, Y.I. Michtom. A structured approach to programming. New Jersey: Prentice Hall Inc., 1977.

2. H. Katzan. Fortran 77. New York: Van Nostrand Reinold Company, 1978.

3. B. Kernighan, P.J. Plauger. The elements of programming style. New York: Yordon Inc., 1978.

4. J.-P. Lamoitier. Exercices de programmation en Fortran IV. Paris: Dunod, 1977.

MIR publicará

Mayorov S., Kirilov V., Pribluda A.
INTRODUCCIÓN A LOS MICROORDENADORES

El libro ilustrado en colores sirve como guía práctica para ingenieros y peritos que no tienen una preparación sistemática en el campo de los microordenadores. Abarca, prácticamente, todas las cuestiones claves del funcionamiento y utilización de los microordenadores. Se examinan estructuras de microordenadores nacionales y extranjeros, lo específico en su funcionamiento y utilización, acción mutua con distintos dispositivos exteriores. El libro ofrece la metodología de la construcción de modelos de microordenadores y las recomendaciones para la organización del "trabajo práctico" con los microprocesadores y microordenadores examinados.

Se recomienda para ingenieros ocupados en la creación y explotación de dispositivos técnicos con microordenadores.

A NUESTROS LECTORES:

Mir edita libros soviéticos traducidos al español, inglés, francés, árabe y otros idiomas extranjeros. Entre ellos figuran las mejores obras de las distintas ramas de la ciencia y la técnica, manuales para los centros de enseñanza superior y escuelas tecnológicas, literatura sobre ciencias naturales y médicas. También se incluyen monografías, libros de divulgación científica y ciencia-ficción.

Dirijan sus opiniones a la Editorial Mir, 1 Rizhski per., 2, 129820, Moscú, I-110, GSP. URSS.